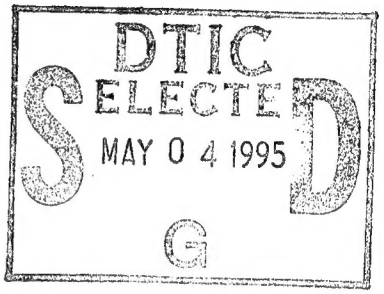
AFIT/GCS/ENG/95M-01



VISAGE: IMPROVING THE BALLISTIC VULNERABILITY MODELING AND ANALYSIS PROCESS

THESIS

Brett F. Grimes

Computer Engineer, 88 CG/SCSA, USAF

AFIT/GCS/ENG/95M-01

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED

4. TITLE AND SUBTITLE

VISAGE: IMPROVING THE BALLISTIC VULNERABILITY MODELING AND ANALYSIS PROCESS

5. FUNDING NUMBERS

6. AUTHOR(S)

Brett F. Grimes, Computer Engineer, 88 CG/SCSA, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology, WPAFB, OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GCS/ENG/95M-01

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Hugh Griffis
ASC/XRESV
WPAFB, OH 45433
255-2353 x2353

Marty Lentz
WL/FIVS
WPAFB, OH 45433
255-6302

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution unlimited;

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The purpose of this thesis was to improve the process of modeling and analyzing ballistic vulnerability data. This was accomplished by addressing two of the more urgent needs of vulnerability analysts; the ability to display fault tree data and to edit target descriptions. A vulnerability data visualization program called VISAGE was modified to meet these needs.

VISAGE was originally created to preview static shotline plots and subsequently grew into a full-featured visualization package for vulnerability target descriptions and analyses data. The next logical step in the programs evolution was to include the needed editing and fault tree display capabilities.

The editing features were divided into basic and advanced categories. The basic editor allows users to manipulate parts at the lowest level of a target description. The advanced editor allows users to manipulate a target description at its highest levels.

The display of fault tree data is a cutting edge feature, since, currently, no other vulnerability package can offer the same capability. The fault tree data is also linked to the target description geometry, so that users can view fault tree data dependencies.

14. SUBJECT TERMS

Vulnerability, Ballistic, Visualization, Editing, Fault Tree, FASTGEN, COVART

15. NUMBER OF PAGES
174

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

AFIT/GCS/ENG/95M-01

VISAGE: IMPROVING THE BALLISTIC VULNERABILITY MODELING AND ANALYSIS PROCESS

THESIS

Presented to the Faculty of the Graduate School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Computer Systems

Brett F. Grimes

Computer Engineer, 88 CG/SCSA, USAF

January 1995

Preface

The purpose of this thesis was to improve the process of modeling and analyzing ballistic vulnerability data. This was accomplished by addressing two of the more urgent needs of vulnerability analysts; the ability to display fault tree data and to edit target descriptions. A vulnerability data visualization program called VISAGE was modified to meet these needs.

I would like to thank all the people who have given me advice, support, and encouragement while writing this thesis. I would like to thank my thesis advisor, Dr. Henry Potoczny, for his insights into Boolean logic. I would also like to thank Lt. Col. Martin Stytz (committee member) and Lt. Col. (Ret.) Patricia Lawlis for their advice and continual patience. My gratitude goes to the sponsors of this thesis, Hugh Griffis and Marty Lentz, for constantly providing me with new challenges. Also, I greatly appreciate the efforts of Capt. Oscar Lessard, who displayed unending skill in discovering "unintended features" in my code. I also appreciate the forbearance and support of my fellow co-workers in 88 CG/SCSA during my attendance at AFIT. Especially, my former supervisor, Dr. Bob Jurick, and my current supervisor, F. Keith Jones, who enthusiastically supported my educational efforts. Finally, I would like to thank my fiancee', Patty and my family, for their continual understanding, love, and support during my endeavors at AFIT.

Brett F. Grimes

Table of Contents

Preface ii

List of Figures vii

List of Tables viii

Abstract ix

I. Introduction 1

Problem Statement 1

Objectives 2

Approach 2

II. Background 4

Survivability 4

Susceptibility 5

Vulnerability 6

Ballistic Vulnerability Assessment Process 6

Modeling 7

FASTGEN 7

Analysis 9

COVART 10

HOOPS Graphics Library 10

Open Software Foundation (OSF) Motif Widgets 11

Push Buttons 12

Toggle/Radio Buttons 12
Labels 12
Menu Bar 12
Pulldown Menus 12
Option Menus 12
Scales 13
Text Fields 13
Scrolled Windows 13
Scrolled Lists 13
Scrolled Text 13
Message Dialogs 13
File Selectors 14
Popup Dialogs (Non-standard) 14
Arrow Buttons w/Text Fields (Non-standard) 14
VISAGE 14
Conclusion 19
III. Methodology 20
Enhancements Tasks 20
Target Description Editing Features 20
Basic Editor 20
Basic Editor Features 21

FASTGEN Bulk Data File (BDF) Organization 21

VISAGE BDF Data Structures 22

Basic Editor Function Implementation 25

Basic Editor GUI Design 26

Advanced Editor 29

Advanced Editor Features 29

Advanced Editor Function Implementation 30

Advanced Editor GUI Design 31

Fault Tree Analysis Background 33

Fault Tree Data Display 35

Intuitive 35

Compact 37

Standard Motif Widgets 37

Interaction 38

Fault Tree Data Format 38

Set Level 39

Function Level 39

System Level 39

Subsystem Level 40

Component Level 40

Fault Tree Data Viewer (FTDV) GUI Design 40

VISAGE Fault Tree Data Structure 43

IV. Enhancement Effort Results 46

VISAGE 46

Pros 46

Cons 46

Structure 46

Target Description Editing Enhancements 47

Fault Tree Data Display Enhancement 49

V. Conclusions and Recommendations 52

Strengths and Limitations 52

Target Description Editing 52

Fault Tree Data Display 52

Practical Implications 53

Recommendations 53

Conclusion 54

Appendix A: Computer Code Routine List (By File) 55

Appendix B: Computer Code Outline 70

Appendix C: Interface Motif Widget Summary 88

Appendix D: VISAGE 2.2 User's Manual 92

Bibliography 172

Vita 174

List of Figures

Figure 1 The Modeling Process 7

Figure 2 The Analysis Process 10

Figure 3 VISAGE Advanced Rendering Example 16

Figure 4 VISAGE Orthogonal Slicer Example 17

Figure 5 BDF Categorization 23

Figure 6 VISAGE BDF Data Structures 25

Figure 7 Basic Editor Preview Capability Example 28

Figure 8 DCE and DV Interfaces 29

Figure 9 Advanced Editor GUI Design 32

Figure 10 Advanced Editor Preview Capability 33

Figure 11 Fault Tree Boolean Logic Operations 34

Figure 12 Example Fault Tree 36

Figure 13 Simple Fault Tree Example 42

Figure 14 Target Description for Example Fault Tree 43

Figure 15 Fault Tree Segment Names and Organization 45

List of Tables

Table 1 BDF Card Categories 23

Table 2 Depth-first Tree Traversal Output 36

Table 3 Breadth-first Tree Traversal Output 37

Abstract

The purpose of this thesis was to improve the process of modeling and analyzing ballistic vulnerability data. This was accomplished by addressing two of the more urgent needs of vulnerability analysts; the ability to display fault tree data and to edit target descriptions. A vulnerability data visualization program called VISAGE was modified to meet these needs.

VISAGE was originally created to preview static shotline plots and subsequently grew into a full-featured visualization package for vulnerability target descriptions and analyses data. The next logical step in the program's evolution was to include the aforementioned needed capabilities.

The editing features were divided into basic and advanced categories. The basic editor allows users to manipulate parts at the lowest level of a target description. Users can modify existing parts or create entirely new ones. The advanced editor allows users to manipulate a target description at its highest levels. Users can then create or modify large pieces of a target description. A previewing feature available on both editors allows users to double-check any modifications or additions.

The display of fault tree data is a cutting edge feature, since, currently, no other vulnerability package can offer the same capability. To simplify the implementation, the fault tree data is displayed in an outline format. There are two main benefits of using the outline format. First, the format is easily represented by the low-level Motif widgets of which the Graphical User Interface (GUI) of VISAGE is constructed. Second, the outline format is a more compact form for displaying the fault tree data.

VISAGE: IMPROVING THE BALLISTIC VULNERABILITY MODELING AND ANALYSIS PROCESS

I. Introduction

The question of vulnerability modeling and analysis is not why it is needed, but of how it can best be accomplished. The need for vulnerability modeling and analysis is readily demonstrated by a number of incidents from the recent Gulf War, for example:

"In one incident, an F-16C was hit by warhead fragments that damaged both ends of the engine. Although the fan, several compressor stages, and the exhaust nozzle were damaged, the engine and afterburner continued to provide thrust, permitting the aircraft to climb rapidly to high altitude and return 500 miles to base [1:33]."

Without the benefit of vulnerability modeling and analysis early on and throughout the design stages of the F-16, that aircraft, and quite possibly the pilot, would most likely have been lost.

Problem Statement

The problem with the ballistic vulnerability modeling and analysis process is basically a lack of the right software tools. Vulnerability analysts have a choice of several programs to create target descriptions of aircraft and to analyze the vulnerability aspects of these aircraft. However, very few software tools exist that allow the analysts to manipulate and correct the structure of FASTGEN-format target descriptions and view the accompanying analysis results. To simplify the ballistic vulnerability modeling and analysis process, a software tool is needed that will address these issues.

Objectives

The first objective of this project was to determine how to best address the lack of software tools. The obvious answer was to look at the currently used tools. A visualization program called VISAGE is widely used to view FASTGEN-format target descriptions and associated analysis data. VISAGE is an Air Force-owned program previously developed by the author for the sponsors of this project, Hugh Griffis of ASC/XRESV and Marty Lentz of WL/FIVS. It was determined that the simplest course of action would be to add the necessary tools to the VISAGE program. The next objective was to determine what features could be added to VISAGE to provide the needed capabilities. Several meetings were held with the sponsors to determine what features would be the most beneficial to the vulnerability assessment process. From these meetings, two features were identified for addition to the VISAGE program:

1. Comprehensive target description editing capabilities.
2. Display of vulnerability fault tree data in conjunction with target descriptions.

Approach

To address the issue of editing capabilities, several more meetings with the sponsors were required to determine the editing features needed by vulnerability analysts. Then it was necessary to reconcile those needs with the program structure of VISAGE and the capabilities of the underlying graphics system. This resulted in the set of editing features that would be added to VISAGE. Given the editing features, the next step was to come up with the appropriate Graphical User Interface (GUI) design.

The next step was to determine how to add the fault tree data display capabilities to VISAGE. This required a study of the theory and uses of Fault Tree Analysis (FTA). Having gained an understanding of the uses of FTA in vulnerability assessments, it was

then necessary to determine the best method of presenting fault tree data to the analyst. After a display format was decided upon, it was then necessary to determine how best to implement the display in terms of the GUI of VISAGE.

Having completed the addition of the enhancement features, the results of the effort needed to be evaluated. This was done through a staged release of the program. After completing the target description editing additions, the new VISAGE program was released to selected test sites for evaluation. The results of these evaluations were then used to correct and refine the enhancements. The same approach was taken for the fault tree data display enhancements.

II. Background

Before addressing the process of ballistic vulnerability modeling and analysis, some background is needed.

Survivability

"Aircraft combat survivability is defined here as the capability of an aircraft to avoid and/or withstand a man-made hostile environment [2:1]." The measure of an aircraft's survivability is based on probabilities. Survivability is denoted by P(S), probability of survival, and is the complement of the probability of a kill. The probability of a kill is composed of two other probability factors; susceptibility and vulnerability. Susceptibility is measured by the probability of a hit, and vulnerability is measured by the probability of a kill given a hit.

Probability of Kill = Susceptibility * Vulnerability

$$P(K) = P(H) * P(K/H) \quad (1)$$

where

P(K) = Probability of Kill

P(H) = Probability of Hit (susceptibility)

P(K/H)= Probability of Kill given a Hit (vulnerability)

The probability of survival, P(S) is given by:

$$P(S) = 1 - P(K) \quad (2)$$

$$= 1 - (P(H) * P(K/H)) \quad (3)$$

Examination of equations (1) and (3) lead to the conclusion that to increase the probability of survival, P(S), the reduction of the susceptibility and/or the vulnerability of the aircraft is required. However, these reductions must not significantly degrade the mission capability of the aircraft.

Susceptibility

Susceptibility is defined as an aircraft's inability to avoid a man-made hostile environment and can be quantified as the probability that an aircraft is hit by a damage-causing mechanism. Susceptibility generally consists of three categories:

1. threat activity
2. detection, identification, and tracking
3. threat engagement

The measure of the susceptibility of an aircraft is dependent upon the probability of a threat being ready to engage, the probability of the aircraft being detected, and the probability of the threat being employed against the aircraft. An aircraft's susceptibility can be influenced by factors such as design (signature reduction, maneuverability, etc.), tactics, and survivability equipment (jammers, chaff, etc.). The reduction of an aircraft's susceptibility is usually accomplished in one or more of the following ways:

1. threat warning
2. electronic jammers and deceivers
3. signature reduction (stealth)
4. expendables
5. threat suppression
6. tactics

Vulnerability

Vulnerability is defined as an aircraft's inability to withstand the damage caused by the man-made hostile environment and can be quantified as the probability of a kill given a hit. This probability is expressed as the ratio of the vulnerable area of the aircraft to the presented area of the aircraft. An aircraft's vulnerable area is influenced by the design (component redundancy, location, etc.) and survivability features (damage suppression, etc.). The reduction of an aircraft's vulnerability is usually accomplished in one or more of the following ways:

1. component redundancy with separation
2. component location
3. passive damage suppression
4. active damage suppression
5. component shielding
6. component elimination

Ballistic Vulnerability Assessment Process

Several software programs can be used to create vulnerability assessments. This thesis focuses on the assessments based on data from the FASTGEN and COVART programs. The ballistic vulnerability assessment process deals with the determination of an aircraft's ability to withstand various ballistic threats, such as small arms, machine gun, and cannon fire. This assessment process is composed of two parts; modeling and analysis.

Modeling

Modeling ballistic threats can be done in several ways. This thesis is concerned with the modeling methods used by FASTGEN. The modeling part of the process consists of the following steps:

1. Aircraft model generation (from CAD, FEA, or other source)
2. Translation of model to FASTGEN target description format (if needed)
3. Validation and correction of target description
4. Generation of LOS data

These steps are shown in Figure 1.

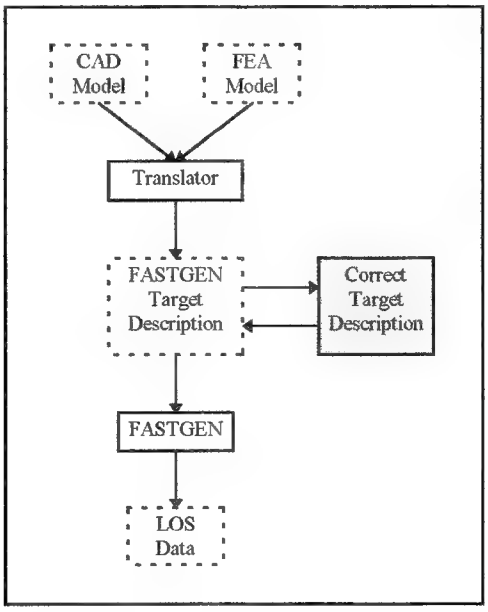


Figure 1 The Modeling Process

FASTGEN

FASTGEN is a program that generates shotlines using a ray-casting technique. Shotlines are the paths followed by ballistic threats. These paths may or may not intersect with a target description. A target description is a geometric model of an aircraft. In

FASTGEN, a target description file is also known as a bulk data file. A target description has three levels of detail; the Group, Component, and Element levels. The Group level is the top level and is divided into twelve types (for aircraft):

1. Aircraft Skin
2. Power Plant
3. Crew
4. Flight Control System
5. Fuel System
6. Ammunition (including bombs)
7. Armament
8. Structural Members
9. Electrical/Avionics Systems
10. Miscellaneous
11. Threat (added for VISAGE)
12. Fragment (added for VISAGE)

These Group level types cover the main functional areas of an aircraft for the purposes of vulnerability analysis. Each Group can consist of up to 999 Components.

The Component level is the middle level and can consist of a maximum of 999 Elements. No special types are assigned to the component level. Generally, components correspond to the systems and subsystems of a target description.

The Element level is the lowest level. Elements are used to describe the basic parts of a target description.

Target descriptions are generally based on Computer-Aided Drafting (CAD) or Finite Element Analysis (FEA) models, but some are created specifically for vulnerability

studies. The CAD and FEA models are then translated into the FASTGEN bulk data format. This transition is not always smooth, since the translation programs may introduce errors. Also, the specifically created models may have errors. To ensure accurate analysis results, the errors in the FASTGEN target description must be corrected. Once the target description is corrected, an analyst can use FASTGEN to create Line-Of-Sight (LOS) data. LOS data is basically a listing of all the components of a target description that are encountered by each shotline. In other words, it is a damage record for each projectile threat. This LOS data can then be used as input data for the COVART program.

Analysis

Analysis of ballistic threats can also be done in several ways. This thesis is mainly concerned with the methods used by COVART. The analysis part of the process consists of the following steps:

1. Generation of COVART input files (specifications for threat type, component vulnerabilities, etc.)
2. Run COVART with input files and LOS data
3. Output of P(K) results
4. Output of fault tree data (optional)
5. Repeat Step 2 for various threats

These steps are shown in Figure 2.

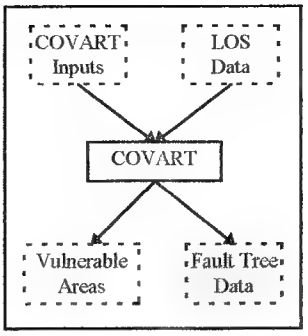


Figure 2 The Analysis Process

COVART

COVART is a program that can be used to evaluate the FASTGEN shotline data for a target description. COVART has several data input files that are used to describe the aircraft in terms of its vulnerabilities. COVART can produce the various probabilities of failure for each component described by the input data files and the LOS data. COVART can also produce a fault tree output file that describes the interdependencies of all the components of a target description. A fault tree is a graphical representation of the Boolean equation of failure for a target description. Generating a fault tree is an important capability because it allows vulnerability analysts to see the weak points of an aircraft. This is key to applying the six methods of vulnerability reduction listed previously. By being able to tell where an aircraft's weaknesses are, an aircraft designer can go back to the design and implement the appropriate vulnerability reduction technique.

HOOPS Graphics Library

HOOPS is a set of graphics routines used for creating interactive graphics applications. HOOPS is, mainly, a database that stores information about the drawing, display, and rendering of objects. The HOOPS database is organized as a tree-shaped

hierarchy, resembling a file directory. The system provides an application program with tools for modifying, querying, searching, and displaying the HOOPS database.

The data consists of geometric primitives, cameras, lights, modelling attributes, and application-specific information. Related elements are grouped together in segments, which are the units of organization within the database. Each segment may also contain other segments, hence the resulting tree structure. The hierarchical grouping is an efficient way of organizing data, since it lets the user manipulate the component parts of objects, or of groups as a whole.

All information stored in the database can be changed. Geometry can be edited, attributes can be modified, and the hierarchy can be reshaped. After a series of changes is completed, the user directs the system to update the display to reflect the new contents of the database. The sequence of determining the user's request, modifying the database accordingly, and redisplaying the image becomes the major theme in an interactive graphics program [3:1].

Open Software Foundation (OSF) Motif Widgets

Motif is the name for the OSF window management system for the X Window System. The X Window System is a window-based display system found mostly on UNIX-based computer systems. Motif provides software developers with a set of X Window-compliant GUI components known as widgets. These widgets provide basic user interface capabilities and some of them can be combined to create more complex widgets, referred to here as compound widgets. The GUI for VISAGE is based on the standard OSF Motif widget set and some compound widgets created specifically for VISAGE. A summary of the Motif widgets used in the interface can be found in Appendix C. The following is a list of the Motif widgets in the interface and how they are used.

Push Buttons

Push buttons are widgets that initiate an action or function when selected ("clicking on") with the mouse (or other input device).

Toggle/Radio Buttons

Toggle buttons essentially act like "check boxes" and may be used to initiate an action or set an option or flag. Radio buttons are a special case of toggle buttons, in that only one out of a group may be selected at any time. The name is derived from the buttons used for selecting preset stations on radios.

Labels

These are text strings that are used for labeling purposes in an interface.

Menu Bar

This is an organizing widget that only contains pulldown menus.

Pulldown Menus

This is a menu, found in menu bars, that the user activates by selecting once or by holding down the mouse button on the selected menu text. This brings up a menu of push buttons or other widgets for the user to select.

Option Menus

Option menus are a special case of pulldown menus, in that they may appear in places other than a menu bar and have a different appearance. This selection becomes the "top" of the "menu" and is displayed on the widget.

Scales

Scale is the Motif name for a sliding bar. This widget lets the user select a number from a preset range of numbers. Users can elect to "drag" the sliding bar to a selected point using the left mouse button, jump to a selected point using the middle mouse button, or increment or decrement the current selected value by clicking to the right or left, respectively, of the sliding bar.

Text Fields

These widgets allow the user to type in text and may also be called "text type-ins".

Scrolled Windows

Scrolled windows allow the user to scroll around a window containing more items than can be displayed at once in the allotted window size.

Scrolled Lists

Scrolled lists allow the user to scroll through a list of choices and to select one item.

Scrolled Text

Scrolled text widgets are used to allow the user to scroll through text information that may be larger than the available display area.

Message Dialogs

Message dialogs are used to confirm an action, such as exiting the program.

File Selectors

File selectors are compound widgets (composed of several different widgets, but treated as one large widget) that allow the user to select files for input or output purposes.

Popup Dialogs (Non-standard)

Popup dialogs are compound widgets that are created by the GUI designer. These dialogs are used to separate different functions of the program, to compartmentalize operations and to prevent information overload.

Arrow Buttons w/Text Fields (Non-standard)

These are compound widgets that are similar in function to the Scales. A user can increment or decrement the value displayed in the associated text field by clicking on either the up or down arrows, respectively. The user can also input a specific value by typing inside the text field.

VISAGE

VISAGE is a liberally derived acronym for **VIS**ualization of Vulnerability **A**nalysis and **GE**ometry. (The title of VULnerability Geometry and Analysis Results (VULGAR) was a close second.) VISAGE is a U.S. Air Force-developed program written in the C programming language, with the graphics display handled by calls to the HOOPS Graphics Library from Ithaca Software. The Graphical User Interface (GUI) for VISAGE is based on Motif, a window management interface to the X Window System.

VISAGE is a specialized program designed to allow vulnerability analysts to visualize, verify, and correct FASTGEN target descriptions and analysis data. It provides users with a combination of visualization and CAD program features. Each feature was

added to address specific requirements of vulnerability analysts. The end result is a program custom-made for working with FASTGEN data. Some important features of VISAGE are:

1. Advanced rendering capabilities
2. Orthogonal slicing of target descriptions
3. View controls (rotation, zoom, and pan)
4. Color contours
5. Animation
6. Hard copy output (PostScript, EPS, HPGL, CGM, and PICT)
7. Motif-based Graphical User Interface (GUI)

The advanced rendering capabilities allows users to determine the visibility, color, and transparency of a target description down to the component level. It also provides lighting with flat, Gouraud, or Phong shading and specular highlighting. The shading is limited to HOOPS "shell" geometry items. Shell geometry items are figures such as hexahedrons, cylinders, or spheres that are created by giving HOOPS a set of points and a connection list (face list) for the points. Non-shell geometry items will be flat-shaded. Figure 3 shows an F-15 with Gouraud shading and the aircraft skin semi-transparent.

The orthogonal slicer provides six cutting planes, with each axis having a cutting plane for the positive and negative directions. Users can activate as many of the six cutting planes as they want and position them anywhere along their respective axes. Figure 4 shows an F15 with a cutting plane applied at the mid-section of its port-side engine.

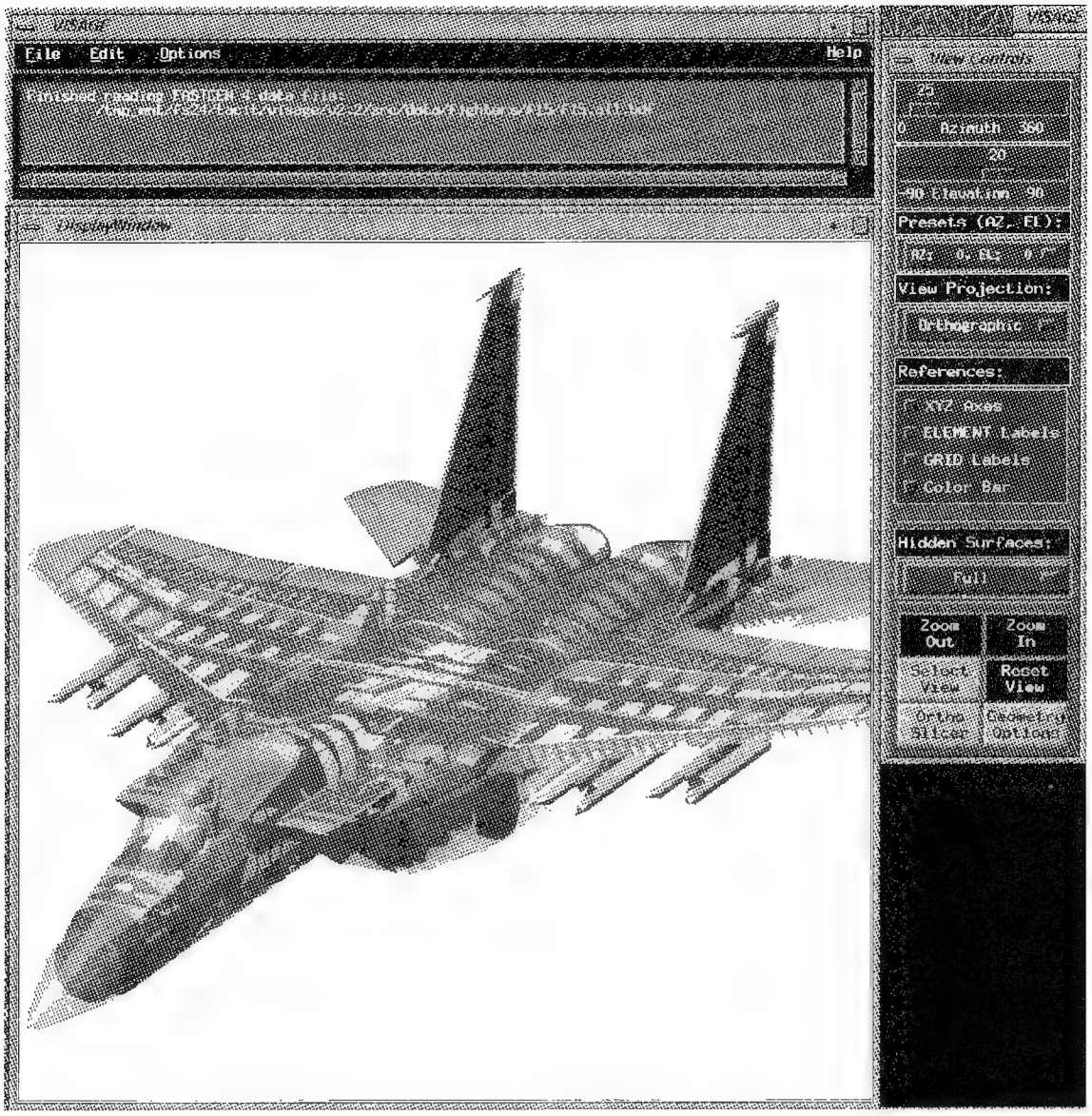


Figure 3 VISAGE Advanced Rendering Example

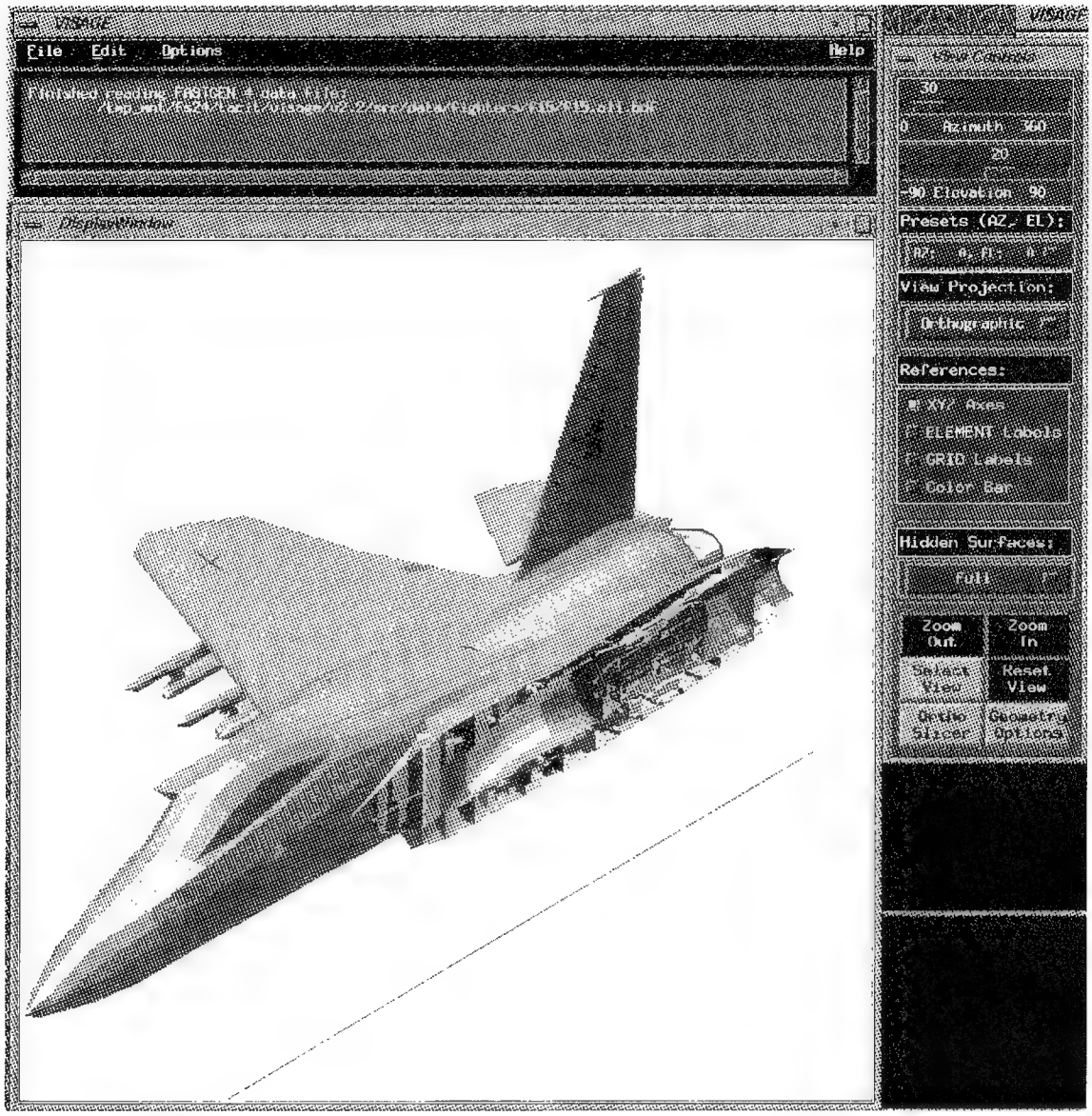


Figure 4 VISAGE Orthogonal Slicer Example

The View Controls dialog allows users to specify specific azimuth and elevation coordinates for the camera viewpoint. A pull-down menu allows users to jump to preset views, as defined in the FASTGEN manual. The View Control dialog also serves as the main controls interface. It can summon other control interfaces, such as the Orthogonal Slicer dialog, the Geometry Options dialog, and the Select View dialog. It contains push-buttons for zoom operations and to reset the display to the default view. Another pull-

down menu allows users to select different hidden surface algorithms. There are also controls to set the visibility of the coordinate axes and the target description's grid and element numbers. The View Controls dialog can be seen at the right side of Figures 3 and 4.

VISAGE allows users to display analysis data as color "contours" on the target descriptions. Through the Contour Data File, users can specify that components and elements of a target description be displayed in any of 30 different colors. This feature is linked to the animation capabilities, so that users can step through changes in the data set.

The animation features are currently limited to the camera viewpoint. Sets of viewpoint data enable users to "fly" the camera view around the "space" occupied by a target description.

Users can also output the currently displayed target description in any of five different formats; PostScript, Encapsulated PostScript, HPGL, CGM, and PICT. In addition, the HPGL and CGM formats allow users to specify the plot in either an A (8.5"x11"), B (11"x17"), C (17"x22"), D (22"x34"), or E (34"x44") size.

The most underrated feature is the Motif-based GUI. Most users today take graphical interfaces for granted and have come to expect them in all software. With GUI building and management tools creating an interface for an application is not nearly as hard as it was in the past. The trickiest part of creating an interface is trying to design in intuitiveness of use. Issues as simple as color schemes can take on great importance when evaluating an interface for its "look and feel".

Conclusion

Ballistic vulnerability analysis is an essential part of combat aircraft design and production. Data collected during various wars and conflicts has demonstrated this repeatedly. The problem has been one of the availability of adequate software tools to perform the necessary analyses. The VISAGE software was created to provide some of the capabilities needed by vulnerability analysts. To further address the modeling and analysis issues of the ballistic vulnerability assessment process, the two features of target description editing and fault tree data display were identified for addition to VISAGE. The remainder of this study will be concerned with the development and addition of these two features.

III. Methodology

This thesis focuses on improving the ballistic vulnerability assessment process by adding needed capabilities to the VISAGE vulnerability data visualization software package. This chapter describes the addition of the target description editing and fault tree data display capabilities.

Enhancements Tasks

Two enhancement tasks were identified:

1. Add comprehensive target description editing features.
2. Add fault tree data display features.

Target Description Editing Features

To simplify the development, the editing features were divided into two categories; basic and advanced. The reason for this was one of scope. The features required by the survivability analysts applied to different levels of the target description. The features identified for the basic editor apply at the Element and Component levels of a target description. The features identified for the advanced editor apply at the Component and Group levels. Thus, all levels of the target description may be edited through the two editors.

Basic Editor

Three steps were followed in adding basic editing features to VISAGE:

1. Determine which features are needed.
2. Determine how the needed features can be implemented in VISAGE.
3. Determine the appropriate GUI interface components.

Basic Editor Features

Four basic features for elements and components were identified:

1. creation
2. deletion
3. editing
4. copying

These four features provide all the functionality needed for a basic editor. Next, it was necessary to decide how to best implement these functions in the context of the FASTGEN bulk data file format and the internal data structures of VISAGE.

FASTGEN Bulk Data File (BDF) Organization

The FASTGEN bulk data file format is basically a listing of the components of a target description. The data entries of the BDF are known as "cards," with each 80-character card consisting of 10 fields of 8 characters each. Most BDF data entries are one card long, but some require a second card, referred to as a continuation card. Each component is designated by a special data entry, known as a SECTION card, which specifies the group to which the component belongs, the identification number of the component, the material type of the component, and whether the component contains plate or volume elements. Each component consists of GRID, POINT (optional), and element cards (e.g., CQUAD, CTRI, CSPHERE, etc.), which immediately follow the SECTION card. The GRID cards specify the 3-dimensional coordinates used by the element cards. The POINT cards are used to specify the coordinate positions of fragments. The element cards are used to specify the geometric entities that make up the target description. An example BDF can be found in Appendix A of the VISAGE 2.2 User's Manual, which is found in Appendix D of this document.

VISAGE BDF Data Structures

The original VISAGE BDF data structure was a dynamically allocated 2-dimensional character array. A subroutine would read the contents of a FASTGEN BDF into this array and, using the contents of the array, another subroutine would build the target description. Then, since the data was no longer needed, the array was deallocated. Adding editing capabilities to the program required that the data be retained and kept accessible. The original data structure could have been retained, but it did not provide the best access to the different components of the BDF. Therefore, the next step was to determine a more suitable data structure, taking into account the organization of the BDF. Examination of the BDF format led to the conclusion that the cards in the BDF could be grouped into 5 categories:

1. global data
2. component data
3. grid data
4. point data
5. element data

The global category is singular, but the component, grid, point, and element categories are repeated for each component in the BDF. The global data category contains all data entries that apply to the complete target description. The component data category holds all cards that supply information about a specific component. The grid and point categories contain all the grid and point cards, respectively, of a component. The element category contains all the element cards of a component. Figure 5 shows an example of the categorization of the BDF.

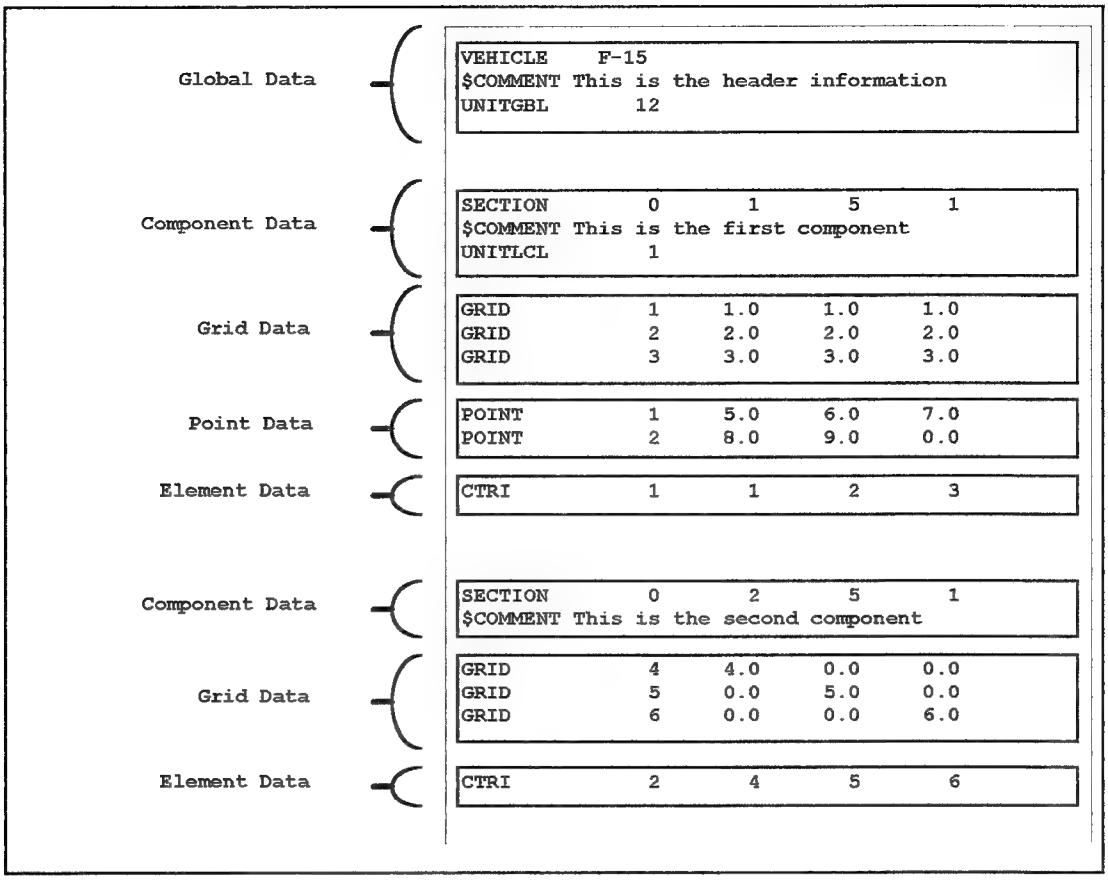


Figure 5 BDF Categorization

Table 1 lists the BDF cards in each category. A complete description of the formats for these cards can be found in the VISAGE 2.2 User's Manual (Appendix E) and the FASTGEN 4.1 User's Manual.

Global	Component	Grid	Point	Element
VEHICLE	SECTION	GRID	POINT	CCONE1
HOLE	UNITLCL			CCONE2
WALL	CORDAE			CHEX1
$COMMENT	$NAME			CHEX2
UNITGBL	$COMMENT			CLINE
				CQUAD
				CSPHERE
				CTRI

Table 1 BDF Card Categories

In VISAGE, these categories are implemented as two structures; one for the global (or header) data and one for the component, grid, point, and element data. The reason for this is that the global data can be kept in one block, while the component-related data must be kept for each component specified in the target description. The global data is kept in a dynamically allocated structure array called **TGheader**. **TGheader** is a pointer of type **dline**. Type **dline** is a structure that consists of one 82-character array, in which one card may be stored. Initially, **TGheader** is allocated one **dline** entry. As global data cards are read in, **TGheader** is allocated more **dline** entries, one per card. The component-related data is kept in a static 2-dimensional array called **TGdata**. **TGdata** is a 12 x 1000 array of type **targetgeom**. Type **targetgeom** consists of four pointers (**comp**, **grid**, **point**, and **elem**) of type **dline**, four integer variables (**ccnt**, **gcnt**, **pcnt**, and **ecnt**) to serve as counters for the arrays pointed to by **comp**, **grid**, **point**, and **elem**, and one integer variable (**update**) that serves as a geometry update flag. Initially, **comp**, **grid**, **point**, and **elem** are null pointers and the integer variables are set to zero. As the cards of each component in the BDF are read in, the **comp**, **grid**, **point**, and **elem** pointers of the specified component are allocated **dline** entries and the entry counts are updated accordingly. The update flag is also set to true. Since the FASTGEN BDF does not require components to be specified in numerical order, the array for the pointers is statically allocated for simplification and to avoid overhead due to dynamic allocation and ordered insertion operations. The dynamic arrays for the individual components are not allocated unless the components are specified in the BDF or are created during an editing session. Figure 6 shows the data structures and the corresponding C programming language declarations.

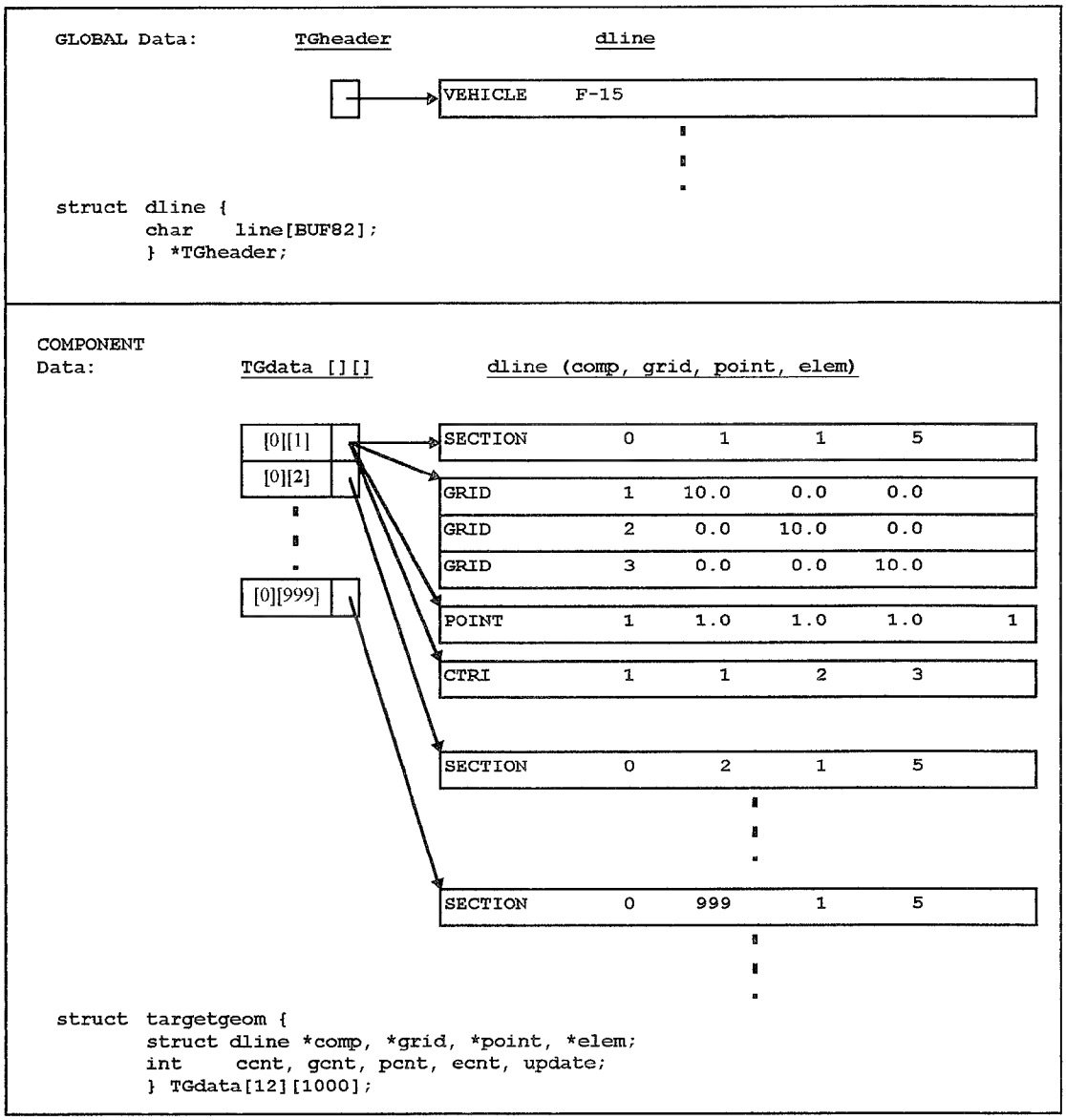


Figure 6 VISAGE BDF Data Structures

Basic Editor Function Implementation

Given the data structures, the next step was to implement the four basic editing functions of creation, deletion, modification and copying. This was accomplished by making some modifications to the subroutine (**read_bdf**) used to input the BDF data into the data structures. It was necessary to modify **read_bdf** to accept input from either a file

or a text buffer associated with the Basic Editor. The four editing operations could then be handled through the Basic Editor code. This would be done by putting the appropriate data entry cards in the text buffer and then calling **read_bdf**. In this way, the user can create new components and modify or copy existing components. The deletion of components is accomplished by deleting all the entries of an existing component and calling **read_bdf**. The **read_bdf** subroutine detects no text in the buffer and frees up the arrays associated with that component and updates the counters.

Basic Editor GUI Design

Given the data structures for the BDF and the implementation of the basic editing functions, the next step was to come up with a GUI design for the Basic Editor. After consultations with my thesis sponsors and selected VISAGE users, it was determined that the most desirable features for the GUI would be:

1. A simple method of editing components and elements
2. A simple method to create new BDF format cards
3. A previewing capability

The first feature was readily taken care of by using an editable scrolled text window widget. Use of this widget allows users to directly edit the elements in one or more components. The widget also provides an internal text buffer that can be accessed by the **read_bdf** subroutine. Another benefit is that the widget provides users with the capability to move or copy text inside its boundaries, making it easy to copy or reorder card entries in a component.

It was determined that the easiest way to implement the second feature would be to create a template mechanism. This template would allow the user to fill in the blanks to create any BDF format card. The design of the template was taken from the BDF card

format pages found in the FASTGEN 4.1 User's Manual and the VISAGE 2.2 User's Manual, so as to present users with a familiar format for creating cards.

Implementing the previewing feature required the ability to evaluate the data from the Basic Editor without adding it to the target description. The actual target description geometry is created by two subroutines; **read_bdf** and **build_geom**. The **read_bdf** subroutine inserts BDF data into the BDF data structures and the **build_geom** subroutine creates the target description based on the contents of the BDF data structures. To preview components without prematurely adding them to the target description, it was necessary to duplicate the **read_bdf** and **build_geom** subroutines and the BDF data structures. The **preview_read** and **preview_geom** subroutines duplicate the capabilities of the **read_bdf** and **build_geom** subroutines, but keep the data in structures separate from **TGheader** and **TGdata** and only access data from the Basic Editor. Therefore, when the Preview button is activated, the **preview_read** and **preview_geom** subroutines are called and the new geometry is displayed along with the current target description. If the component being previewed already exists, then the target description version is made invisible and the Basic Editor version is displayed instead. The previewed geometry is also displayed with black polygonal faces and red edges to distinguish it from the rest of the target description. Figure 7 shows an example of the previewing feature.

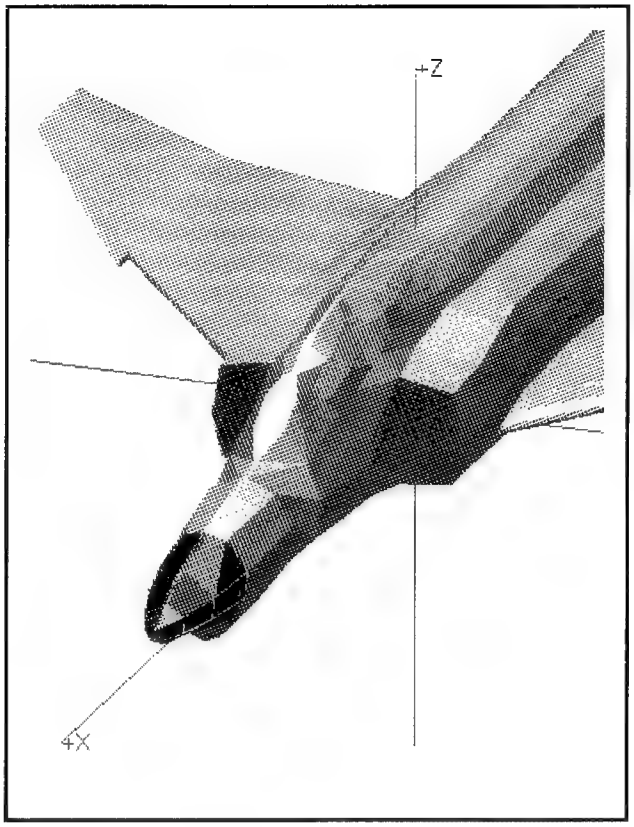


Figure 7 Basic Editor Preview Capability Example

The Basic Editor consists of the Data Card Editor (DCE) and the Data Viewer (DV). The DCE provides the template capability and the DV provides the direct editing and previewing capabilities. Figure 8 shows the DCE and DV interfaces. The DCE was created using label widgets for the field descriptions, text field widgets for the fill-in sections, a pull-down menu widget to select the BDF card type, radio button widgets to select the Header, New, or Existing components to edit, text field widgets to specify the Group and Component, and push-button widgets to Apply or Cancel changes. The DV was created using a scrolled text window widget for the component data, push-button widgets for the Delete (highlighted selection), Delete All (entire component), Undo Delete, and Preview functions, and push-button widgets for the OK, Apply, Reset, and Cancel functions. The functions of all the DCE and DV interface widgets are described in Section 2.9 of the VISAGE 2.2 User's Manual in Appendix D.

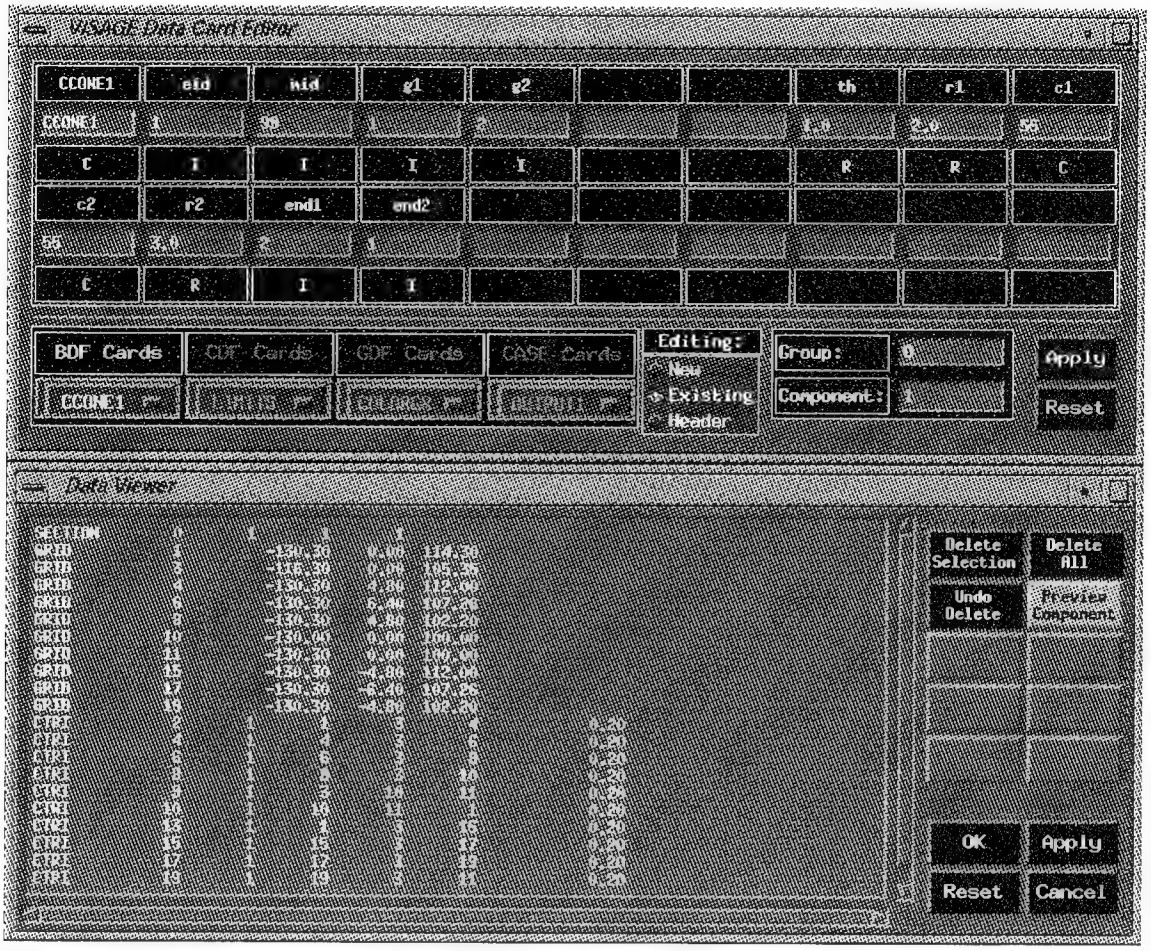


Figure 8 DCE and DV Interfaces

Advanced Editor

The same three steps used in creating the basic editor were followed in adding advanced editing features to VISAGE.

Advanced Editor Features

Six advanced editing features for groups and components were identified as needed:

1. rotation
2. scaling
3. translation

4. mirroring
5. copying
6. deletion

These six features provide all the basic functionality needed for the Advanced Editor. Each function can be applied singly or in combinations (except deletion) to groups or components of a target description.

Advanced Editor Function Implementation

The implementation of the copy and deletion features was made easier by the VISAGE BDF data structures. The data structures allow the copy function to be a simple matter of array duplication. The user selects the source component(s) to be copied and the destination component(s) and the source arrays are copied into the destination arrays. The copy function will only be performed if the number of destination components matches the number of source components. If a destination component already exists or a source component does not exist, the copy operation will skip that component and continue. Since the basic editor can only delete one component at a time, the deletion function is also provided in the Advanced Editor to allow users to delete ranges of components or even entire groups. The data structures make the deletion function a simple matter of freeing up the dynamic arrays associated with the deleted components and resetting the array counters to zero.

HOOPS provides complete matrix manipulation routines as part of its geometry manipulation capabilities. Using these matrix operations allows a custom transformation matrix to be assembled from the values specified in the Advanced Editor. The user can create a transformation matrix by specifying the values for any combination of the rotation, scaling, translation, and mirroring functions. The limitations are that the functions can only be used once in a transformation and that they are applied in the following order:

rotation, scaling, translation, and mirroring. Also the user must specify a reference point for the rotation and scaling operations, otherwise the origin (0,0,0) will be used by default.

Advanced Editor GUI Design

It was determined that the most desirable features for the GUI would be:

1. A simple method of applying the six advanced editing features
2. A previewing capability

Since the method of applying the six features was determined in the implementation phase, the next step was to design the appropriate interface elements for each feature. The design of the GUI consists of six parts:

1. Group and Component range
2. Reference Point
3. Rotate/Scale/Translate operations
4. Mirror/Copy operations
5. Delete and Preview operations
6. OK/Apply/Reset/Cancel functions

The input interface for the Group and Component range was implemented using text widgets (also referred to as type-ins). The Reference Point input interface was implemented using text widgets for the X, Y, and Z coordinates. The Rotate, Scale, Translate, Mirror, and Copy operations all need to be independently selectable, so each is equipped with a toggle button widget for activation. If the toggle button is not activated, editing of the operation interface is disabled and the operation will not be performed. The Rotate, Scale, and Translate interfaces are implemented using text widgets. The Rotate interface requires Roll, Pitch, and Yaw inputs. The Scale and Translate interfaces require

X-axis, Y-axis, and Z-axis inputs. The Mirror and Copy operations are linked because the Mirror operation must create a new component for the mirrored copy. The Copy function can be performed by itself. The Mirror/Copy interface is composed of three toggle buttons that are used to indicate which coordinate planes (YZ, ZX, or XY) around which to mirror the component(s). The YZ, ZX, and XY planes correspond to the X, Y, and Z axes, respectively. The Copy interface is implemented using text widgets and is a duplicate of the Group and Component range input interface. Push-button widgets are used for the Delete, Preview, OK, Apply, Reset, and Cancel functions. The functions of all the Advanced Editor interface widgets are described in Section 2.10 of the VISAGE 2.2 *User's Manual* in Appendix D. Figure 9 shows the GUI design of the Advanced Editor.

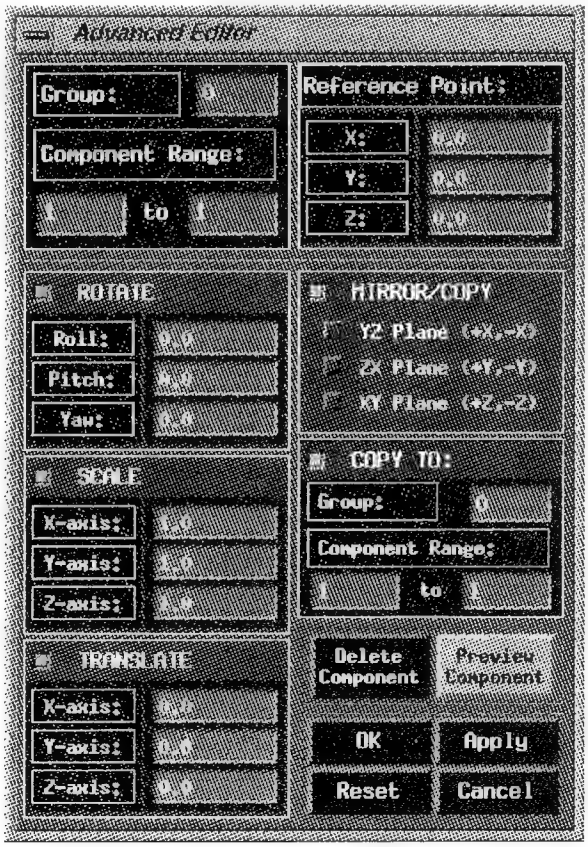


Figure 9 Advanced Editor GUI Design

Implementing the previewing feature in the Advanced Editor was much simpler than in the Basic Editor. Since the geometry is already present, it was only necessary to copy the appropriate components into a temporary segment and apply the specified transformations. The previewed geometry is displayed with black polygonal faces and red edges to distinguish it from the rest of the target description. Also, the original components are made invisible to avoid conflicts or confusion. When the user is finished previewing, the temporary segment is removed and the original components are restored. Figure 10 shows an example of the Advanced Editor preview capability.

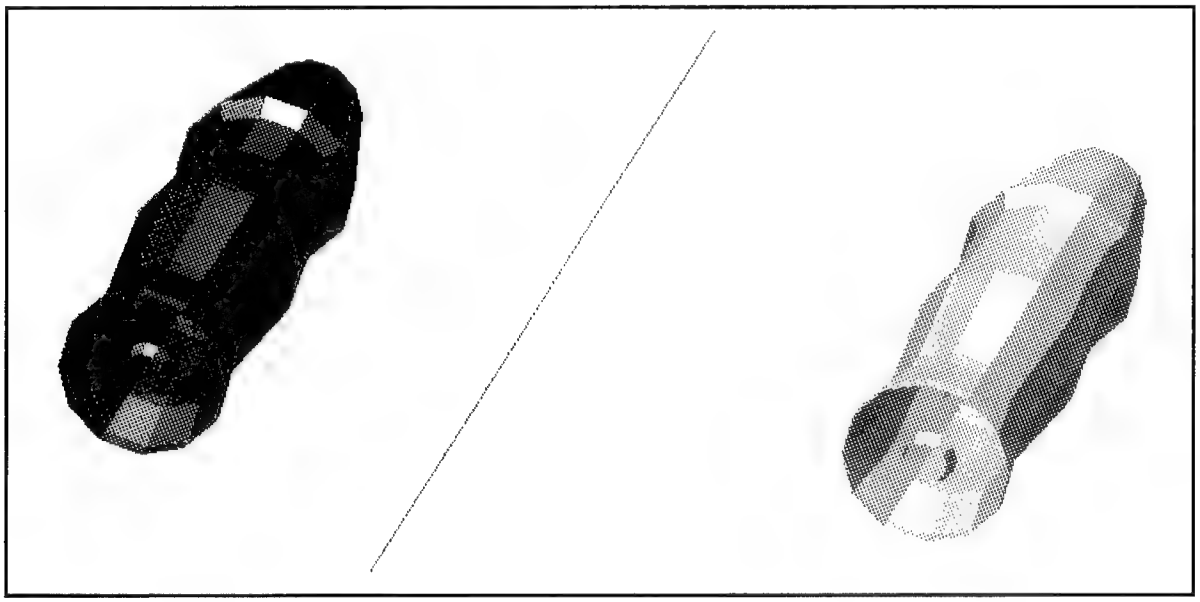


Figure 10 Advanced Editor Preview Capability

Fault Tree Analysis Background

Fault Tree Analysis (FTA) is a fairly recent analysis approach. It was developed in 1961 by Bell Telephone Laboratories to evaluate the Minuteman Launch Control System and was presented to the public in 1965 in Seattle [5:29]. FTA relies on deductive reasoning and was developed as an alternative to tabular Failure Modes and Effects Analysis (FMEA) and its inductive approach. FMEA is a "bottom-up" technique where all

subsystem. Put more simply, a failure of a component is assumed and all the possible consequences of that failure are mapped out in the FMEA table. FTA takes a "top-down" approach, starting with a failure event (fault) and branching out to the preceding events that caused the fault. The relations between these faults are described by Boolean logic equations.

The fault tree is a graphical representation of the Boolean equation of the failure events. A fault tree consists of elements arranged in serial and/or parallel fashion. The serial and parallel arrangements are equivalent to the Boolean OR and AND operations, respectively. In a simple serial arrangement, at least one element must fail for the entire construct to fail (Figure 11a.). In a simple parallel arrangement, one element in each path must fail for the entire construct to fail (Figure 11b.). A hybrid arrangement, containing both serial and parallel constructs, will fail according to the combined effect of all the constructs (Figures 11c).

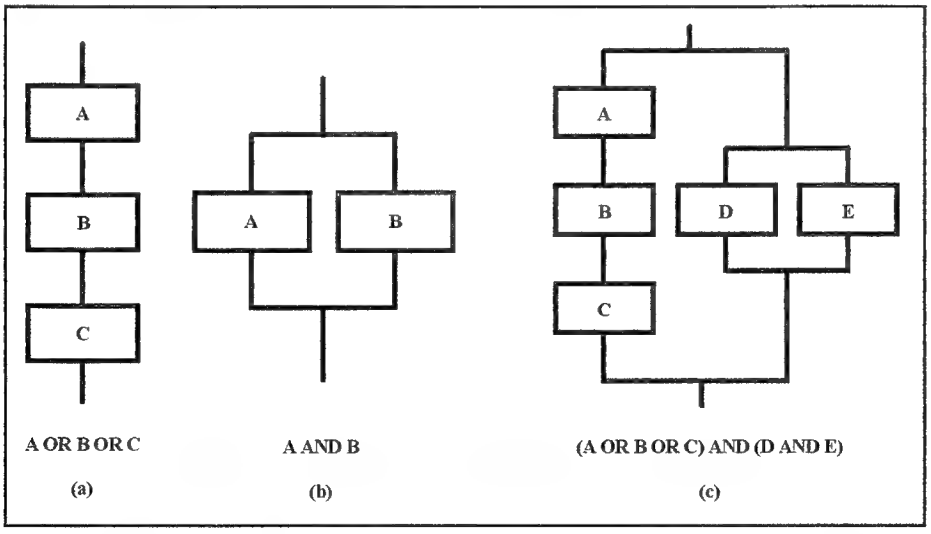


Figure 11 Fault Tree Boolean Logic Operations

The usual graphical representation of a fault tree is a block diagram as seen in Figures 11a-c. An advantage of this is that the relationships between elements are readily seen. A disadvantage is that, for large fault trees, the representation may not be completely

displayed on a computer screen and may take many pages to print. Another drawback is that Motif currently does not have a standard widget for displaying data in a tree format. Therefore, it was necessary to determine a display format capable of being represented by combinations of low-level Motif widgets while still adequately conveying the tree-like nature of the data. These problems are addressed by the VISAGE fault tree display features.

Fault Tree Data Display

To display the fault tree data, four requirements must be taken into account:

1. The display must be as intuitive as possible.
2. The output should be as compact as possible.
3. The display must be built using standard Motif widgets.
4. The display must be capable of interacting with the target description.

To simply fulfill these requirements, an outline display format was selected. The outline format is basically a row-column matrix. Each row contains only a single entry and the column position indicates the level of the entry in the tree. The organization of the entries in the outline comes from a depth-first traversal of the fault tree the outline represents.

Intuitive

Tree diagrams are easy to interpret because they present the hierarchical and relational aspects of data sets in a simple format. In a tree diagram it easy to see the relationships between elements and how they combine to form larger entities. A tree diagram starts at a root node and branches out into lower levels. Each level down represents a greater degree of detail. The outline format represents this data in a similar fashion, except that each branch is listed down to

its end (or leaf) nodes, as in a depth-first traversal of the tree. A breadth-first traversal could have been used to provide the outline entries, but the relationships between the various levels would have been less intuitive. Figure 12 shows an example fault tree and Tables 2-3 show examples of depth-first and breadth-first tree traversal output. The Boolean operation associated with each entry is listed beside the entry name in the tables.

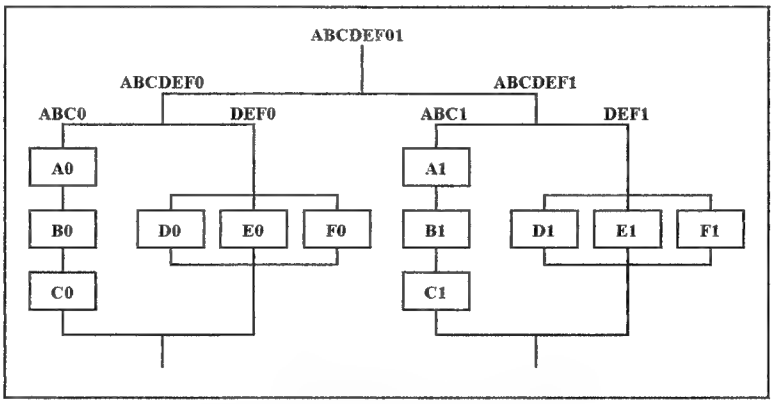


Figure 12 Example Fault Tree

ABCDEF01		
ABCDEF0	<AND>	
ABC0	<AND>	
A0		<OR>
B0		<OR>
C0		<OR>
DEF0	<AND>	
D0		<AND>
E0		<AND>
F0		<AND>
ABCDEF1	<AND>	
ABC1	<AND>	
A1		<OR>
B1		<OR>
C1		<OR>
DEF1	<AND>	
D1		<AND>
E1		<AND>
F1		<AND>

Table 2 Depth-first Tree Traversal Output

ABCDEF01	
ABCDEF0	<AND>
ABCDEF1	<AND>
ABC0	<AND>
DEF0	<AND>
ABC1	<AND>
DEF1	<AND>
A0	<OR>
D0	<AND>
E0	<AND>
F0	<AND>
A1	<OR>
D1	<AND>
E1	<AND>
F1	<AND>
B0	<OR>
B1	<OR>
C0	<OR>
C1	<OR>

Table 3 Breadth-first Tree Traversal Output

Compact

While a tree format is very intuitive, hardcopy output can become quite large. Hardcopy output of large tree diagrams is also limited by current printer technology. Currently, large tree diagrams must either be printed using large pen or electrostatic plotters or be printed out in strips on line printers and assembled by hand. Miniaturization is possible, but can result in an unacceptable loss of detail. The outline format solves this problem because the output consists of lines of text. This allows the user to output the diagram on almost any printer.

Standard Motif Widgets

The requirement for building the display with standard Motif widgets was an issue of portability. Standard Motif widgets were needed to be able to guarantee that the interface would work across all supported platforms. Creating a

custom widget to display tree diagrams would have increased the potential for problems in the future. Changes in Motif or the platform operating systems could have potentially rendered the widget inoperative or faulty. By using standard Motif widgets, compliance and interoperability was assured across all supported platforms.

Interaction

The last requirement was that the interface be able to interact with the target description. This was needed to allow the user to view the relationships between the components listed in the tree data, as well as with the rest of the target description.

Fault Tree Data Format

The fault tree data format is the implementation of the outline format. The data format has a hierarchical nature as a result of the outline format's basis in the depth-first tree traversal. In other words, a data file is composed of several sections, each one representing a pass of the depth-first tree traversal algorithm. This also results in a strict ordering of the data format that allows the fault tree to be constructed as a data file is input. Basically, the format consists of five conceptual levels:

1. Set Level
2. Function Level
3. System Level
4. Subsystem Level
5. Component Level

Each level represents a corresponding part of a fault tree. In order to accommodate fault trees more than five levels deep, the Subsystem level is recursive. For

very shallow trees, the System and Subsystem levels can be omitted. The following paragraphs explain the function of each level. A complete description of the data cards implementing each level can be found in Section 3.5 of the VISAGE 2.2 User's Manual in Appendix D.

Set Level

The Set Level consists of one or more sets specified by FTSET cards. The FTSET card allows the user to specify a name for the set and how many functions it contains. Each set represents the top level of a particular tree. Generally, there will only be one set (or tree) per input file, but there is no limitation on the number of sets.

Function Level

The Function Level consists of a number of functions specified by FTFUNC cards. The FTFUNC card allows the user to specify a name for the function, how many systems it contains, what Boolean operation links the functions, an optional failure field that specifies the number of failures needed for a fault out of the total elements in an AND operation, and a color identification number for highlighting purposes. Each function represents the top level of a particular function being modeled in the tree.

System Level

The System Level consists of a number of systems specified by FTSYS cards. The FTSYS card allows the user to specify a name for the system, how many subsystems it contains, what Boolean operation links the systems, and an optional failure field that specifies the number of failures needed for a fault out of the total elements in an AND operation.

Subsystem Level

The Subsystem Level consists of a number of subsystems specified by FTSUB cards. The FTSUB card allows the user to specify a name for the subsystem, how many subsystems it contains, what Boolean operation links the subsystems, and an optional failure field that specifies the number of failures needed for a fault out of the total elements in an AND operation. The FTSUB card is unique in that it is the only card that can have other cards of the same type under it, i.e., the FTSUB cards can be nested several layers deep, while the FTSET, FTFUNC, and FTSYS cards can only appear on their respective levels.

Component Level

The Component Level consists of a number of components specified by FTCOMP and $CNAME cards. The FTCOMP card allows the user to specify a name for the component, how many target description components it contains, and what Boolean operation links the components. The $CNAME card allows the user to specify the target description group and component numbers, the P(K) table identification number, the kill type, and a 40-character text description of the component. The FTCOMP card is really a special case of the FTSUB card and is used to identify the ends (or leaf nodes) of a particular branch of the tree. These leaf nodes are designated by $CNAME cards. An FTCOMP card always precedes one or more $CNAME cards and may take the place of the FTSUB card in branches where the FTSUB card would be redundant.

Fault Tree Data Viewer (FTDV) GUI Design

The dynamic nature of the construction of the FTDV is a unique feature in VISAGE. Each time a fault tree data file is read in, the FTDV must create a whole new set

of push-buttons. The rest of the interfaces in VISAGE are all created at program start-up and are hidden when not being used.

The design of the FTDV was pre-determined by the outline format. Since the outline format consists of text entries and the entries must have the capability to interact with the target description geometry, push-buttons were considered the best choice of widgets for the interface. Push-buttons can display text and, through their activation callbacks, can be used to interact with the geometry. A scrolled window widget was selected to contain the push-button entries, for cases when the fault tree is larger than the viewer display area. A push-button was also used for the Reset Display function, which resets any highlighted geometry to its original state.

Given the outline format and the selected Motif widgets, the next step was to implement a representation of the outline format in the FTDV GUI. Each entry in the format is represented by a push-button. The appearance of each push-button is determined by the type of outline entry it represents. The color of the entry indicates the associated Boolean operation. Red indicates an AND operation, blue indicates an OR operation, and black indicates no associated operation.

All push-buttons are created the same height. This allows the coordinate position of the push-buttons to be used as an index into the fault tree data structure. When a push-button is selected, an X Window utility function (XGetWindowAttributes) provides the Y coordinate of the push-button. This is divided by the push-button width and the resulting quotient is the index of the corresponding entry in the fault tree data structure array. The fault tree data structure is described in the next section.

In the outline format, each row may contain only one entry, which is assigned a width based on its type. All outline entries are of equal width except those representing the $CNAME cards. The $CNAME push-buttons must be able to display the Group,

Component, and 40-character description of the entry. The FTFUNC, FTSYS, and FTSUB entry push-buttons must have room to display the Name of the entry and, optionally, the failure count described in the Fault Tree Data Format section. The FTSET and FTCOMP entry push-buttons must have room to display the Name of the entry. For simplicity and readability, it was decided to make the push-buttons for the FTSET, FTFUNC, FTSYS, FTSUB, and FTCOMP all the same width. The $CNAME entry push-buttons were made five times wider to be able to display the larger amount of information.

Figure 13 shows the Fault Tree Data Viewer with a simple fault tree example and Figure 14 shows the corresponding target description. In Figure 13, the white outline around the red entry at the upper left indicates that the geometry associated with that part of the fault tree is currently being highlighted in the Display Window.

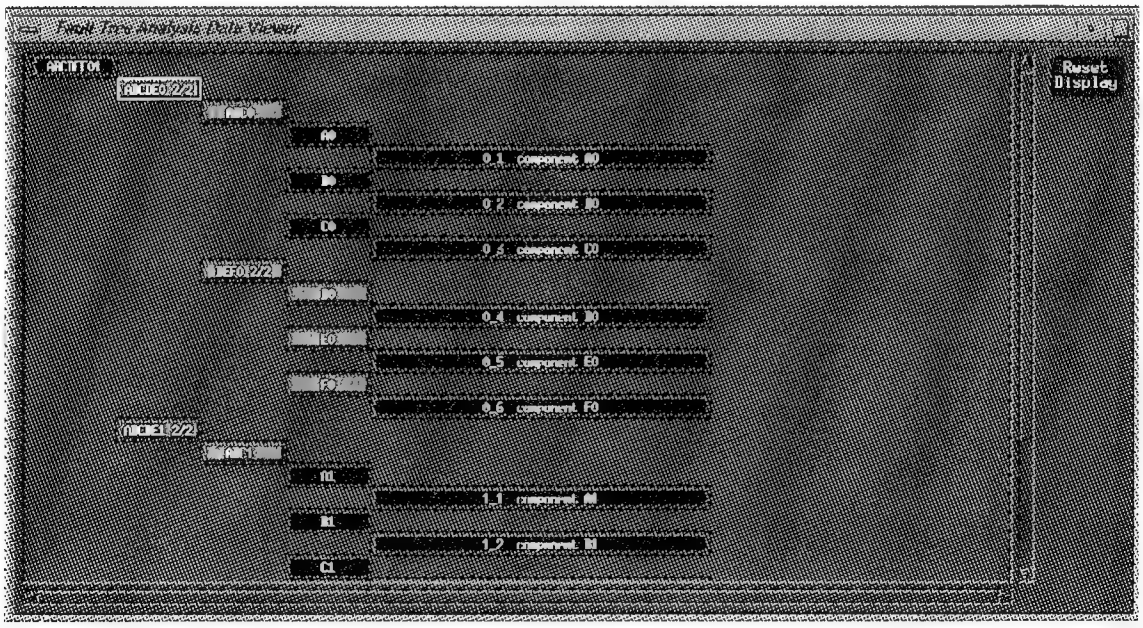


Figure 13 Simple Fault Tree Example

The components corresponding to the selected section of the fault tree are colored blue on the aircraft shown in Figure 14.

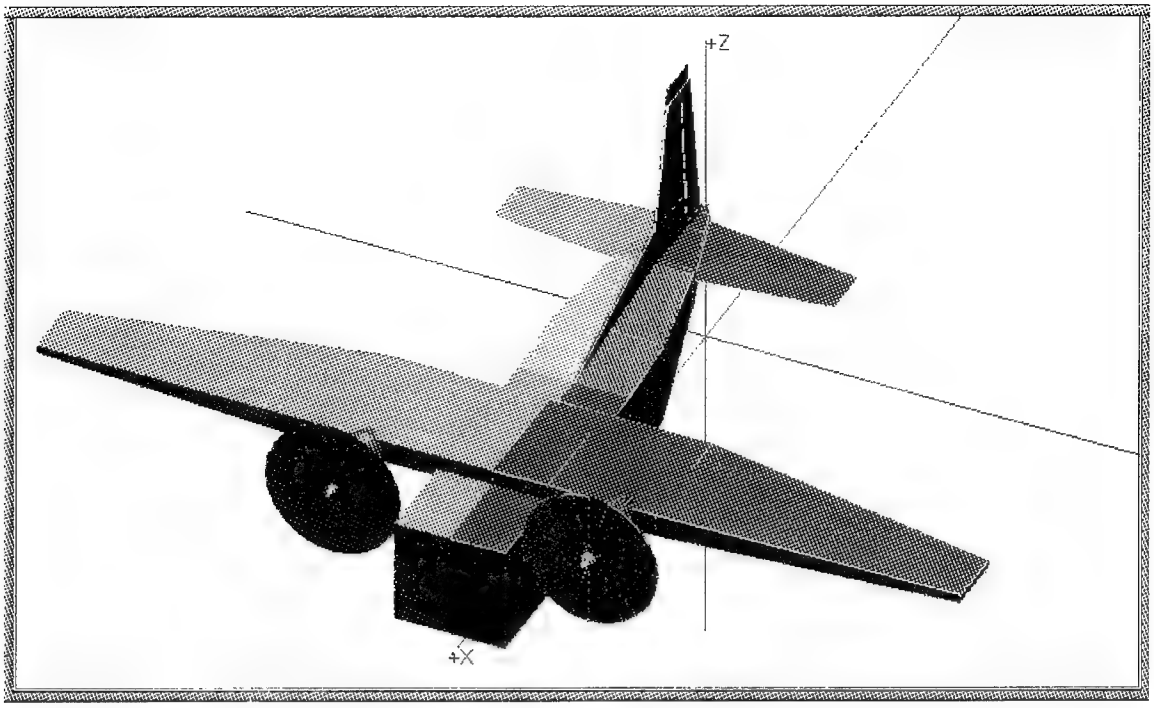


Figure 14 Target Description for Example Fault Tree

VISAGE Fault Tree Data Structure

The C language data structure for the fault tree data is shown below:

```
struct  FTAData {
            Widget      ewidget;
            char        ename[BUF64], esegment[BUF256];
            } *FTADVentry;
```

As seen above, the data structure array for the fault tree data is composed of three parts:

1. The widget variable name
2. The fault tree output text string
3. The segment name variable

The widget variable name is assigned when a push-button is created for a fault tree entry. This variable name is needed when adding the activation callback to the push-

button. When a new Fault Tree Data File (FDF) is input , the widget variable name is needed to destroy the push-button, so the new push-buttons can be created in the FTDV.

The fault tree output text string is the text that is written out to the fault tree output file. In the case of the FTSET and FTCOMP entries, the text string is composed of the entry name and the associated Boolean operation. For the FTFUNC, FTSYS, and FTSUB entries, the text string is composed of the entry name, the failure field described previously, and the associated Boolean operation. The $CNAME entry text string is composed of the Group and Component numbers, the 40-character component description, a P(K) table identification number, and a kill type classification. The fault tree output file is a hardcopy version of the outline format of the fault tree currently being displayed in the FTDV.

The segment name variable holds the name of the HOOPS segment of the current entry. Since only the $CNAME entries actually reference the target geometry, it was necessary to set up a HOOPS segment sub-tree that would allow non-$CNAME entry push-buttons to highlight the components associated with the levels below them. The easiest solution for this was to create a segment for each non-$CNAME entry push-button using the current array index as the segment name. Then, the $CNAME entries are added at the lowest level of the HOOPS segment tree and, due to the hierarchical inheritance of HOOPS segments, can be highlighted from the upper levels. Figure 15 shows the HOOPS segment names and entry numbers for an example fault tree. The "?plot" string in the segment names is the name of the HOOPS segment that contains the target description geometry. The "/" character in the segment names is a HOOPS convention indicating the inheritance ordering of the segments, similar to UNIX pathnames. The "x_x" strings are a VISAGE convention of using the combined Group and Component numbers as a segment name.

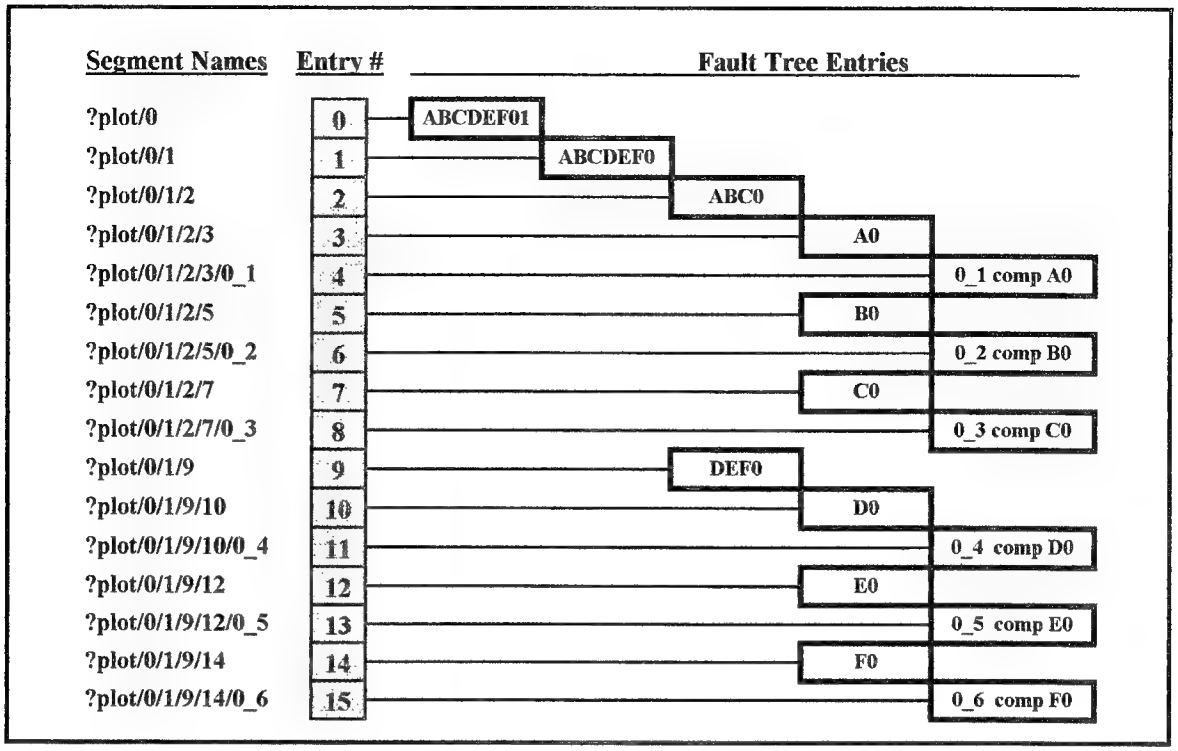


Figure 15 Fault Tree Segment Names and Organization

IV. Enhancement Effort Results

The focus of this thesis was to add editing and fault tree display capabilities to the VISAGE software program. This section discusses the results of those efforts. Since there were no experimental results, this section will comment on the enhancement efforts.

VISAGE

Pros

VISAGE originally provided vulnerability analysts with the ability to visualize FASTGEN-format target descriptions and associated analysis data. It provided these capabilities across several platforms and at no cost to the user, since it is a USAF-developed software package. It provides these capabilities through advanced graphics and a relatively simple Motif-based GUI.

Cons

One of the drawbacks to the VISAGE program is that the graphics performance is not equal across all platforms. Another is that it is dependent upon a non-industry standard graphics library. In terms of this project, it did not already provide full support for the features that had been chosen to be added.

Structure

The VISAGE program consists of three linked parts:

1. The internal data structures and application code.
2. The HOOPS graphics elements.
3. The Motif-based GUI.

The internal data structures and application code are the basis of the program. The HOOPS graphics elements and Motif-based GUI form the display for VISAGE. In order to allow for potential changes, the program is modularized as much as possible. Therefore, if the program should ever need to be implemented with another graphics library or GUI, the changes should not effect the majority of the application code. This modularization also meant that VISAGE was readily modified to support the needed enhancement features. The separation of the internal data structures, the HOOPS graphics elements, and the Motif-based GUI allowed the enhancements to be implemented without a radical redesign of the program.

Target Description Editing Enhancements

In terms of the VISAGE program, the addition of the target description editing capabilities was highly successful. All features required by the project sponsors were added to VISAGE without disruption of the program's basic integrity. By substituting a new internal data structure for storing BDF data, it was possible to easily add the editing features. This data structure change provided an efficient method of storing the target description data, while only requiring slight modifications to the data input and geometry creation subroutines. Use of Motif for the GUI also allowed for the easy addition of interface elements for the basic and advanced editing features. The transformation matrix creation and manipulation capabilities of the HOOPS Graphics Library also provided needed support for the advanced editing features.

Prior to this project, vulnerability analysts had no programs capable of editing a native FASTGEN target description. With the model editing enhancements successfully added to VISAGE during this project, vulnerability analysts now have an editing capability that does not require that data sets be translated into a CAD package data format. The

difficulty in adding the editing capabilities was one of balance. A balance was needed between the analysts' requirements and the complexity of the programming effort. On one hand, the analysts needed certain minimum capabilities to effectively correct target descriptions. On the other hand, the editing interface needed to be simple to avoid straying into areas already adequately covered by existing CAD software packages.

A combination of a template interface and a text editor were decided upon to provide the basic editing capabilities. This avoided the effort of trying to duplicate mouse-based CAD features while simply providing the necessary functionality. The target description elements can be directly edited in the Data Viewer (text editor) and users can also easily create new elements with the Data Card Editor (template interface). Through the combination of these two interface elements, users can create, edit, copy, and/or delete components and elements of a target description.

The analysts also required some advanced editing features to make target description correction easier. The analysts wanted to be able to rotate, scale, translate, mirror, copy, and/or delete parts of a target description. These features are easily applied at the Component and Group levels of a target description. This allows analysts to easily manipulate large sections of target descriptions at one time.

With the addition of the Basic and Advanced Editors to VISAGE, analysts are provided complete target editing description capabilities. Without these features the analysts must edit the target descriptions by hand or translate them into a format usable by some CAD package. Editing the target descriptions by hand can be an unpleasant and daunting task. Most target description files are tens or hundreds of thousands of lines long and often are inadequately documented. Correcting the larger files by hand could take several man-weeks or man-months of effort. The alternative of editing the target description with a CAD package is not a much better choice. This requires not only

translating the target description into another format, but also translating it back into the FASTGEN format, thus doubling the possibility of errors being introduced into the data.

The editing features have been tested by three different groups working on two different projects and have been found to satisfy the vulnerability analysts' needs. The VISAGE editing capabilities have successfully been used to correct defects in the F/A-18 and B-1B FASTGEN target descriptions. Use of the VISAGE editing capabilities on other aircraft target descriptions is planned. The editing capabilities have been reviewed by the sponsors of this project, Hugh Griffis of ASC/XRESV and Marty Lentz of WL/FIVS, and have been pronounced "excellent".

Fault Tree Data Display Enhancement

The other purpose of this thesis was to provide vulnerability analysts with the capability to view fault tree data along with a corresponding FASTGEN target description. In the course of my research into the topic of Fault Tree Analysis (FTA), I found ample support for FTA as an important facet of vulnerability analysis, but very few software tools. This was amazing considering that FTA has been around since 1961. FTA is also widely used for safety analysis studies, but, again, very few software tools appear to be available. Because of this lack, the VISAGE fault tree data display capability represents an important advance in vulnerability analysis.

In terms of the VISAGE program, the addition of the fault tree data display capabilities was more of a study in GUI design than anything else. Since the fault tree data was not integral to the target description, it was only necessary to take advantage of "hooks" into the target description geometry provided by the hierarchical structure of HOOPS. Given these links to the target description, the issue of how to best present the fault tree data was the next challenge. The design of the interface was then just a matter of

providing an efficient solution based on the sponsors' requirements and the features of the standard Motif GUI elements.

To satisfy the survivability analysts' requirements, the fault tree data display needed to be intuitive, compact, composed of standard Motif widgets, and able to interact with target descriptions. The use of the outline format allowed the display to be intuitive and compact. Creating the display with Motif push-button widgets provided the standard and interactive characteristics. The fault tree data display interface meshed well with the VISAGE fault tree data structure. X Window and Motif utility functions also allowed the interface and the data structure to be easily linked together.

Having decided upon the outline format for the fault tree data display and how to implement the Motif interface, the next task was to create a fault tree data file format. COVART currently has support for the input of fault tree data, but the format is complex and the data entries are spread across multiple input files. This makes it difficult not only to create a COVART fault tree data file, but also to input the data into programs such as VISAGE. Therefore, it was decided to create a new fault tree data format; one that could be produced by COVART.

Fault tree data is organized hierarchically. A fault tree data set represents an aircraft in terms of functions, systems, subsystems, and components. Each of these corresponds to different levels in a fault tree. Representing these levels more simply is the purpose of the outline format. The VISAGE fault tree data format is the result of the conversion of the fault tree representation to the outline representation. This conversion is accomplished through the depth-first tree traversal algorithm. This algorithm converts the levels of the fault tree format into the corresponding levels in the outline format. These levels are then represented by the entries in the VISAGE fault tree data format.

The VISAGE fault tree format has been submitted to the COVART development team and is expected to be incorporated into the next release of the product. The format is also being used by personnel in ASC/XRESV for documentation in support of current vulnerability studies. The VISAGE fault tree data format and the Fault Tree Data Viewer have been examined and approved by the sponsors of this project. The sponsors described the VISAGE fault tree capabilities as being an impressive and important new tool for vulnerability analysts.

V. Conclusions and Recommendations

Strengths and Limitations

Target Description Editing

The editing capabilities added to VISAGE during this project have four major strengths:

1. The Basic and Advanced editors allow vulnerability analysts to visualize and verify FASTGEN target descriptions with one software package.

2. The Basic Editor provides users with easy access to all Component and Element data in a target description.

3. The Advanced Editor provides users with the ability to easily create complex components.

4. The editors allow users to preview any changes before they are applied to the target description.

The editors also have two major limitations:

1. The Basic Editor currently only supports the FASTGEN BDF format.

2. Transformations in the Advanced Editor can only be applied in a specified sequence and individual transformations can only be used once in each application.

Fault Tree Data Display

The fault tree data display capabilities added to VISAGE have four major strengths:

1. The Fault Tree Data Viewer provides users with an intuitive and compact interface for viewing fault tree data in an outline format.

2. The FTDV allows users to view the fault tree data in conjunction with the rest of a target description.

3. The VISAGE fault tree input data format is well-structured, yet flexible.

4. The VISAGE fault tree output data format provides users with a concise listing of fault tree data sets.

The fault tree data display capabilities have two major limitations:

1. Fault tree data sets cannot be created or edited with the FTDV interface.

2. Due to CRT screen sizes, the FTDV and the VISAGE Display Window have some overlap.

Practical Implications

The added editing capabilities mean that survivability analysts now have a method of verifying and correcting FASTGEN target descriptions without having to use several different software packages or translators. The editing capabilities are also tailored to the FASTGEN data format, so users do not have to be concerned about finding appropriate and compatible software tools.

The fault tree data display capabilities give vulnerability analysts a much-needed tool for understanding vulnerability study results. Fault trees focus on the critical parts of an aircraft, and being able to see those parts in relation to the rest of the aircraft can be extremely useful. The flexibility of the fault tree data input format and the display interface also allow for the display of non-fault tree data sets. A user can create data files that delineate an aircraft's major and minor systems and subsystems. These files can serve to illustrate design or safety issues, or even be used as documentation for reports.

Recommendations

The VISAGE program now provides survivability analysts with several important tools for conducting ballistic vulnerability analyses. To further enhance the effectiveness of VISAGE, the following items are recommended for further work:

1. The editing and fault tree limitations listed above should be addressed.

2. In order to improve graphics performance while maintaining portability, the feasibility of transitioning the underlying graphics system of VISAGE from the HOOPS graphics library to the OpenGL graphics library should be determined.

3. Since VISAGE is based on earlier coding efforts by the author, a more thorough effort to bring the code up to current Software Engineering standards could be undertaken.

4. The capability to display Line-Of-Sight data sets should be incorporated into VISAGE.

5. To improve model appearance, the mesh resolution of the cone and sphere geometry elements should be dynamically determined according to the relative size of the elements.

Conclusion

The VISAGE software package represents an important link in the ballistic vulnerability analysis process. By combining the properties of CAD and visualization packages, VISAGE allows survivability analysts to bridge the gap between the generation of target descriptions and the generation of analysis data. VISAGE serves as a feedback mechanism by allowing analysts to view target descriptions and analysis data simultaneously. VISAGE also makes a significant contribution to vulnerability analysis by providing one of the few available tools for displaying fault tree data. With the conclusion of this project, VISAGE now stands as a significant achievement in the improvement of the ballistic vulnerability process.

Appendix A: Computer Code Routine List (By File)

****

VISAGE files

vis_display.c

void UpdateDisplay (mpick)
void ResetView ()
void SetBusyPointer (sbpflag)
void SetGeomDispOps ()
void SetCuttingPlanes ()
void PlayAnimation (animcont)
void SelectView (svmanflag, tmindex)
void time_style_segment (tmindex)
void SelectGeom(segment, key, geomtype)
void Mouse (button)
void mouse_cam (xwin_next, ywin_next)

Bool TestEvent (display, event, type)
int Keyboard (keysym, key)

vis_misc.c

void change_key (long origkey, long elemID, int elemType, char *Scope)
void PrintHardcopy ()
void selcolor (segname, grpnum)
void selcontcol (segname,csnum,scolor,gtype,gpattern)
void plotinfo (secnum,cmpnum,eidnum,eidtypind)
UNUSED void build_boxview_seg ()
UNUSED void box_view ()
UNUSED void rubber_box (x0,y0,x1,y1)
UNUSED void Get_String (context,string)

int unpack_key (long key, long *elemID, long *elemType)

vis_FTA.c

void rdFTA (ftaname)

int validftakey ()

vis_rdcontour.c

void rdcontour (cfname)

int validcdkey ()

vis_rdfastgen.c

void rdfastgen (dfile)
void build_geom (group, comp, Eval_Comp)
void build_cylcone (targetseg, r1,r2,th1,th2,e1flag,e2flag)
void build_sphere (targetseg, cntrpt, r1, th)
void camcalc ()
void change_coords (grpnum,compnum,segname)
void init_labels ()
void insert_elemlabels (grpnum, compnum, current)
void maxel_err (gnum,cnum,currcard,current)
void maxmin(x, y, z)
void seg_setup (segname, eid, chkeid, currsnum, curelcnt)
void setaxes ()
void preview_geom (showgeom)
void compute_transform (tmatrixptr)
void copy_components (CompGroup,CompFirst,CompLast,
void transform_components (Group,First,Last,Matrix)
void preview_transform (prevOn,Group,First,Last)
void delete_components (Group, First, Last)

int read_bdf (DataFile, FromFile)
int get_points (loopcnt,currcard,typpnts,current)
int validElemData (dkey)
int validCompData (dkey)
int validHeaderData (dkey)
int cardContinued (dkey)
int validkey (dkey)
int extract_line (currpos, prevbuffer)
int preview_read ()

vis_rdgrpdef.c

void rdgrpdef (gfname)

int validgdkey ()

vis_wininit.c

void wininit ()
void colorbar (topseg)
void def_pointer ()
void def_CB_colors ()
void def_mat_colors ()

vis_wrfastgen.c

void write_file (FileType, FileName)

****

Motif interface files

visage2.c

main (argc,argv)

VISAGE.c

static Widget _Uxbuild_VISAGE()
Widget create_VISAGE()

void postmesg (mesg, printout)
static void activateCB_menu1_File_open()
static void activateCB_menu1_File_save()
static void activateCB_menu1_File_saveas()
static void activateCB_menu1_File_print()
static void activateCB_menu1_File_exit()
static void activateCB_menu1_Edit_Basic()
static void activateCB_menu1_Edit_Advanced()
static void valueChangedCB_menu1_Options_IndepScale()

```
static  void    valueChangedCB_menu1_Options_GeomSelOutput()
static  void    valueChangedCB_menu1_Options_FTCompDisp()
static  void    valueChangedCB_menu1_Options_FTOutline()
static  void    activateCB_menu1_Help_on()

char    *extract_string (cs)
```

viewCtlsbBD.c

```
static Widget   _Uxbuild_viewCtlsbBD()
Widget  create_viewCtlsbBD()

ResetViewControls()
ResetGeomControls()
static  void    focusCB_viewCtlsbBD()
static  void    activateCB_hSurfs_b1()
static  void    activateCB_hSurfs_b2()
static  void    activateCB_hSurfs_b3()
static  void    valueChangedCB_refAxestB()
static  void    valueChangedCB_labelELEMtB()
static  void    valueChangedCB_labelGRIDtB()
static  void    valueChangedCB_colorBartB()
static  void    activateCB_zoomOut()
static  void    activateCB_zoomIn()
static  void    activateCB_selectView()
static  void    activateCB_resetView()
static  void    activateCB_orthoSlicerpB()
static  void    activateCB_geomOpspB()
static  void    activateCB_viewType_Ortho()
static  void    activateCB_viewType_Persp()
static  void    valueChangedCB_azimuthSc()
static  void    valueChangedCB_elevationSc()
static  void    activateCB_presetAZEL_0_0()
static  void    activateCB_presetAZEL_45_0()
static  void    activateCB_presetAZEL_90_0()
static  void    activateCB_presetAZEL_135_0()
static  void    activateCB_presetAZEL_180_0()
static  void    activateCB_presetAZEL_225_0()
static  void    activateCB_presetAZEL_270_0()
static  void    activateCB_presetAZEL_315_0()
static  void    activateCB_presetAZEL_0_45()
static  void    activateCB_presetAZEL_45_45()
static  void    activateCB_presetAZEL_90_45()
```

static void activateCB_presetAZEL_135_45()
static void activateCB_presetAZEL_180_45()
static void activateCB_presetAZEL_225_45()
static void activateCB_presetAZEL_270_45()
static void activateCB_presetAZEL_315_45()
static void activateCB_presetAZEL_0_n45()
static void activateCB_presetAZEL_45_n45()
static void activateCB_presetAZEL_90_n45()
static void activateCB_presetAZEL_135_n45()
static void activateCB_presetAZEL_180_n45()
static void activateCB_presetAZEL_225_n45()
static void activateCB_presetAZEL_270_n45()
static void activateCB_presetAZEL_315_n45()
static void activateCB_presetAZEL_0_90()
static void activateCB_presetAZEL_0_n90()

---

geomDispOpsbBD.c

---

static Widget _Uxbuild_geomDispOpsbBD()
Widget create_geomDispOpsbBD()

ResetSelGroups()group, state, action)
SetGroupInfo()group)
ResetGeomDispOps ()
setgeomcolwin (gcolor)
static void focusCB_geomDispOpsbBD()
static void activateCB_geomDispOpsOKpB()
static void activateCB_geomDispOpsResetpB()
static void activateCB_geomDispOpsApplypB()
static void activateCB_geomDispOpsCancelpB()
static void valueChangedCB_groupNametF()
static void valueChangedCB_grouptB0()
static void valueChangedCB_grouptB1()
static void valueChangedCB_grouptB2()
static void valueChangedCB_grouptB3()
static void valueChangedCB_grouptB4()
static void valueChangedCB_grouptB5()
static void valueChangedCB_grouptB6()
static void valueChangedCB_grouptB7()
static void valueChangedCB_grouptB8()
static void valueChangedCB_grouptB9()
static void valueChangedCB_grouptB10()
static void valueChangedCB_grouptB11()

```
static void   valueChangedCB_compFirsttF()
static void   valueChangedCB_compLasttF()
static void   losingFocusCB_compLasttF()
static void   exposeCB_geomColorSeldA()
static void   singleSelectionCB_geomColorSelsL()
static void   activateCB_gVisibility_b1()
static void   activateCB_gVisibility_b2()
static void   activateCB_gVisibility_b3()
static void   valueChangedCB_transparencySc()
static void   activateCB_gShading_b1()
static void   activateCB_gShading_b2()
static void   activateCB_gShading_b3()
static void   activateCB_gShading_b4()
```

animationCtlsbBD.c

```
static Widget   _Uxbuild_animationCtlsbBD()
Widget  create_animationCtlsbBD()

ResetAnimControls()
static void   focusCB_animationCtlsbBD()
static void   activateCB_playReversedB1()
static void   activateCB_playForwarddB()
static void   activateCB_resetAnimdB1()
static void   activateCB_playStopdB()
static void   activateCB_tCycle_b1()
static void   activateCB_tCycle_b2()
static void   activateCB_tCycle_b3()
static void   activateCB_stepIncDecaB()
static void   activateCB_stepIncIncaB()
static void   activateCB_stepInctF()
static void   activateCB_currStepDecaB()
static void   activateCB_currStepIncaB()
static void   activateCB_currSteptF()
```

orthoSlicerbBD.c

```
static Widget   _Uxbuild_orthoSlicerbBD()
Widget  create_orthoSlicerbBD()

ResetOrthoSlicer()
static void   focusCB_orthoSlicerbBD()
```

static void valueChangedCB_positiveXtB()
static void valueChangedCB_negativeXtB()
static void valueChangedCB_positiveYtB()
static void valueChangedCB_negativeYtB()
static void valueChangedCB_positiveZtB()
static void valueChangedCB_negativeZtB()
static void valueChangedCB_positiveXtF()
static void valueChangedCB_negativeXtF()
static void valueChangedCB_positiveYtF()
static void valueChangedCB_negativeYtF()
static void valueChangedCB_positiveZtF()
static void valueChangedCB_negativeZtF()
static void activateCB_orthoSLicerOKpB()
static void activateCB_orthoSlicerApplypB()
static void activateCB_orthoSlicerResetpB()
static void activateCB_orthoSlicerCancelpB()

---

selectViewbBD.c

---

static Widget _Uxbuild_selectViewbBD()
Widget create_selectViewbBD()

static void focusCB_selectViewbBD()
static void valueChangedCB_cameraPosXtF()
static void valueChangedCB_cameraPosYtF()
static void valueChangedCB_cameraPosZtF()
static void valueChangedCB_cameraTarXtF()
static void valueChangedCB_cameraTarYtF()
static void valueChangedCB_cameraTarZtF()
static void valueChangedCB_viewAngleSc()
static void activateCB_selViewOKpB()
static void activateCB_selViewResetpB()
static void activateCB_selViewApplypB()
static void activateCB_selViewCancelpB()

---

printOpsbBD.c

---

static Widget _Uxbuild_printOpsbBD()
Widget create_printOpsbBD()

SetPOFileType()pofiletype, state)
SetPOSize()size, state)

```
SetPOOrientation()orient, state)
SetPOOutput()pooutput, state)
ResetPrintOps ()

static  void    focusCB_printOpsbBD()
static  void    valueChangedCB_printPStB()
static  void    valueChangedCB_printEPStB()
static  void    valueChangedCB_printHPGLtB()
static  void    valueChangedCB_printCGMtB()
static  void    valueChangedCB_printPICTtB()
static  void    valueChangedCB_printAtB()
static  void    valueChangedCB_printBtB()
static  void    valueChangedCB_printCtB()
static  void    valueChangedCB_printDtB()
static  void    valueChangedCB_printEtB()
static  void    valueChangedCB_printLandtB()
static  void    valueChangedCB_printPorttB()
static  void    valueChangedCB_printPrintertB()
static  void    valueChangedCB_printFiletB()
static  void    valueChangedCB_printFiletF()
static  void    valueChangedCB_printPrintertF()
static  void    valueChangedCB_printNCopytF()
static  void    activateCB_printOKpB1()
static  void    activateCB_printResetpB()
static  void    activateCB_printCancelpB()
```

==========

fileOpenOpsbBD.c

==========

```
static Widget   _Uxbuild_fileOpenOpsbBD()
Widget  create_fileOpenOpsbBD()

void    ResetReadFileOps (reploption, state)
static  void    valueChangedCB_openFTOpsBDFtB()
static  void    valueChangedCB_openFTOpsCDFtB()
static  void    valueChangedCB_openFTOpsGDFtB()
static  void    valueChangedCB_openFTOpsFDFtB()
static  void    activateCB_openFTOpsReadpB()
static  void    activateCB_openFTOpsCancelpB()
static  void    valueChangedCB_openFTOpsRAlltB()
static  void    valueChangedCB_openFTOpsAddOtB()
static  void    valueChangedCB_openFTOpsAddNOtB()
```

helpWindowbBD.c

static Widget _Uxbuild_helpWindowbBD()
Widget create_helpWindowbBD()

void SelectHelpFile (whichmsg)
static void activateCB_pushButton1()

confirmErroreD.c

static Widget _Uxbuild_confirmErroreD()
Widget create_confirmErroreD()

static void cancelCB_confirmErroreD()
static void helpCB_confirmErroreD()
static void okCallback_confirmErroreD()

confirmExitqD.c

static Widget _Uxbuild_confirmExitqD()
Widget create_confirmExitqD()

static void cancelCB_confirmExitqD()
static void helpCB_confirmExitqD()
static void okCallback_confirmExitqD()

confirmFCLosswD.c

static Widget _Uxbuild_confirmFCLosswD()
Widget create_confirmFCLosswD() Widget _UxUxParent)

static void okCallback_confirmFCLosswD()

confirmWarningwD.c

static Widget _Uxbuild_confirmWarningwD()
Widget create_confirmWarningwD() Widget _UxUxParent)

static void cancelCB_confirmWarningwD()

```
static  void    okCallback_confirmWarningwD()
```

DisplayWindow.c

```
static Widget   _Uxbuild_DisplayWindow()
Widget  create_DisplayWindow()

void    setup_GL()gl_widget) [SGI CONDITIONAL CODE]
void    test_overlay ()          [SGI CONDITIONAL CODE]
void    get_best_visual (display, visual_depth, visual_type, visual_info)
static  void    createCB_DisplayWindow()
static  void    exposeCB_DisplayWindow()
static  void    inputCB_DisplayWindow()
static  void    resizeCB_DisplayWindow()
```

---

openBDFfSBD.c

```
static Widget   _Uxbuild_openBDFfSBD()
Widget  create_openBDFfSBD()

static  void    cancelCB_openBDFfSBD()
static  void    helpCB_openBDFfSBD()
static  void    okCallback_openBDFfSBD()
```

---

openCDFfSBD.c

```
static Widget   _Uxbuild_openCDFfSBD()
Widget  create_openCDFfSBD()

static  void    cancelCB_openCDFfSBD()
static  void    helpCB_openCDFfSBD()
static  void    okCallback_openCDFfSBD()
```

---

openFDFfSBD.c

```
static Widget   _Uxbuild_openFDFfSBD()
Widget  create_openFDFfSBD()

static  void    cancelCB_openFDFfSBD()
static  void    helpCB_openFDFfSBD()
```

static void okCallback_openFDFfSBD()

openGDFfSBD.c

static Widget _Uxbuild_openGDFfSBD()
Widget create_openGDFfSBD()

static void cancelCB_openGDFfSBD()
static void helpCB_openGDFfSBD()
static void okCallback_openGDFfSBD()

viewDCEDatabBD.c

static Widget _Uxbuild_viewDCEDatabBD()
Widget create_viewDCEDatabBD()

void ResetDataViewer ()
void SetDataViewer (mode)
static void focusCB_viewDCEDatabBD()
static void activateCB_dvDeleteSelpB()
static void activateCB_dvDeleteAllpB()
static void activateCB_dvUndoDeletepB()
static void activateCB_dvPreviewComppB()
static void activateCB_sparepB1()
static void activateCB_sparepB2()
static void activateCB_sparepB3()
static void activateCB_sparepB4()
static void activateCB_sparepB5()
static void activateCB_sparepB6()
static void activateCB_viewDCEDataOKpB()
static void activateCB_viewDCEDataApplypB()
static void activateCB_viewDCEDataResetpB()
static void activateCB_viewDCEDataCancelpB()

bdfCardsbBD.c

static Widget _Uxbuild_bdfCardsbBD()
Widget create_bdfCardsbBD()

void GetCardText ()
void UpdateDataViewer ()
void SetCardType (CardType)

```
void   ResetCardType ()
static void   focusCB_bdfCardsbBD()
static void   valueChangedCB_textField1()
static void   losingFocusCB_textField1()
static void   valueChangedCB_textField2()
static void   losingFocusCB_textField2()
static void   valueChangedCB_textField3()
static void   losingFocusCB_textField3()
static void   valueChangedCB_textField4()
static void   losingFocusCB_textField4()
static void   valueChangedCB_textField5()
static void   losingFocusCB_textField5()
static void   valueChangedCB_textField6()
static void   losingFocusCB_textField6()
static void   valueChangedCB_textField7()
static void   losingFocusCB_textField7()
static void   valueChangedCB_textField8()
static void   losingFocusCB_textField8()
static void   valueChangedCB_textField9()
static void   losingFocusCB_textField9()
static void   valueChangedCB_textField10()
static void   losingFocusCB_textField10()
static void   valueChangedCB_textField11()
static void   losingFocusCB_textField11()
static void   valueChangedCB_textField12()
static void   losingFocusCB_textField12()
static void   valueChangedCB_textField13()
static void   losingFocusCB_textField13()
static void   valueChangedCB_textField14()
static void   losingFocusCB_textField14()
static void   valueChangedCB_textField15()
static void   losingFocusCB_textField15()
static void   valueChangedCB_textField16()
static void   losingFocusCB_textField16()
static void   valueChangedCB_textField17()
static void   losingFocusCB_textField17()
static void   valueChangedCB_textField18()
static void   losingFocusCB_textField18()
static void   valueChangedCB_textField19()
static void   losingFocusCB_textField19()
static void   valueChangedCB_textField20()
static void   losingFocusCB_textField20()
static void   activateCB_bdfCardsApplypB()
static void   activateCB_bdfCardsResetpB()
```

```
static  void    activateCB_bdfCards_ccone1()
static  void    activateCB_bdfCards_ccone2()
static  void    activateCB_bdfCards_chex1()
static  void    activateCB_bdfCards_chex2()
static  void    activateCB_bdfCards_cline()
static  void    activateCB_bdfCards_cquad()
static  void    activateCB_bdfCards_csphere()
static  void    activateCB_bdfCards_ctri()
static  void    activateCB_bdfCards_grid()
static  void    activateCB_bdfCards_hole()
static  void    activateCB_bdfCards_section()
static  void    activateCB_bdfCards_vehicle()
static  void    activateCB_bdfCards_wall()
static  void    activateCB_bdfCards_comment()
static  void    activateCB_bdfCards_name()
static  void    activateCB_bdfCards_cordae()
static  void    activateCB_bdfCards_point()
static  void    activateCB_bdfCards_unitgbl()
static  void    activateCB_bdfCards_unitlcl()
static  void    activateCB_cdfCards_comment()
static  void    activateCB_gdfCards_comment()
static  void    activateCB_caseCards_comment()
static  void    valueChangedCB_dceGrouptF()
static  void    activateCB_dceComptF()
static  void    valueChangedCB_dceComptF()
static  void    valueChangedCB_dceEditNewtB()
static  void    valueChangedCB_dceEditExistingtB()
static  void    valueChangedCB_dceEditHeadertB()
```

---

advancedEditorbBD.c

```
static Widget   _Uxbuild_advancedEditorbBD()
Widget  create_advancedEditorbBD()

ResetAdvancedEditor()
void    ApplyTransforms ()
void    SetScaleVal (svaltext)
static  void    focusCB_advancedEditorbBD()
static  void    valueChangedCB_advEdGrouptF()
static  void    valueChangedCB_advEdCompFirsttF()
static  void    losingFocusCB_advEdCompFirsttF()
static  void    valueChangedCB_advEdCompLasttF()
static  void    losingFocusCB_advEdCompLasttF()
```

```
static  void    valueChangedCB_rotatetB()
static  void    valueChangedCB_rotateXtF()
static  void    valueChangedCB_rotateYtF()
static  void    valueChangedCB_rotateZtF()
static  void    valueChangedCB_scaletB()
static  void    valueChangedCB_scaleXtF()
static  void    losingFocusCB_scaleXtF()
static  void    valueChangedCB_scaleYtF()
static  void    losingFocusCB_scaleYtF()
static  void    valueChangedCB_scaleZtF()
static  void    losingFocusCB_scaleZtF()
static  void    valueChangedCB_translatetB()
static  void    valueChangedCB_translateXtF()
static  void    valueChangedCB_translateYtF()
static  void    valueChangedCB_translateZtF()
static  void    valueChangedCB_refPointXtF()
static  void    valueChangedCB_refPointYtF()
static  void    valueChangedCB_refPointZtF()
static  void    valueChangedCB_mirrortB()
static  void    valueChangedCB_yzPlanetB()
static  void    valueChangedCB_zxPlanetB()
static  void    valueChangedCB_xyPlanetB()
static  void    valueChangedCB_copytB()
static  void    valueChangedCB_copyGrouptF()
static  void    valueChangedCB_copyFirsttF()
static  void    valueChangedCB_copyLasttF()
static  void    losingFocusCB_copyLasttF()
static  void    activateCB_advancedEditorDeletepB()
static  void    activateCB_advancedEditorPreviewpB()
static  void    activateCB_advancedEditorOKpB()
static  void    activateCB_advancedEditorApplypB()
static  void    activateCB_advancedEditorResetpB()
static  void    activateCB_advancedEditorCancelpB()
```

FTADataViewerbBD.c

```
static Widget   _Uxbuild_FTADataViewerbBD()
Widget  create_FTADataViewerbBD()

void    change_seg_attrib (segname, attrib, pattrib)
void    activateCB_FTADVentries (wgt, cd, cb)
void    AddFTADVEntry (ftaindex, xpos, ypos, etype, eqcolor)
static  void    focusCB_FTADataViewerbBD()
```

```
static  void    createCB_FTADVform()
static  void    activateCB_FTADVClosepB()
```

writeBDFfSBD.c

```
static Widget   _Uxbuild_writeBDFfSBD()
Widget  create_writeBDFfSBD()

static  void    cancelCB_writeBDFfSBD()
static  void    helpCB_writeBDFfSBD()
static  void    okCallback_writeBDFfSBD()
```

Appendix B: Computer Code Outline

This outline lists the routines as they are diagramed by the CodeCenter software design tool. Each tab stop indicates the next call level of the program. Unused and architecture-dependent routines have been annotated. All routines containing the letters "Ux" are either UIM/X utility routines or are part of the interface construction. Global routines and expurgated routines have been marked according to the legend.

Legend:

==> : global routine

+++ : routine contains calls to other routines which have been previously shown

```
int main()
        void *create_VISAGE()
                void *_Uxbuild_VISAGE()
                        void activateCB_menu1_File_open()
                                UxPopupInterface(fileOpenOpsbBD)
                        void activateCB_menu1_File_save()
                                void write_file()
                        void activateCB_menu1_File_saveAs()
                                UxPopupInterface(write_BDFfSBD)
                        void activateCB_menu1_File_print()
                                UxPopupInterface(printOpsbBD)
                        void activateCB_menu1_File_exit()
                                UxPopupInterface(confirmExitqD)
                        void activateCB_menu1_Edit_Basic()
                                UxPopupInterface(bdfCardsbBD)
                                        UxPopupInterface(viewDCEDatabBD)
                        void activateCB_menu1_Edit_Advanced()
                                UxPopupInterface(advancedEditorbBD)
                        void valueChangedCB_menu1_Options_IndepScale()
                                ==> void postmesg()
                        void valueChangedCB_menu1_Options_GeomSelOutput()
                                ==> void postmesg()
                        void valueChangedCB_menu1_Options_FTCompDisp()
                                ==> void postmesg()
                        void valueChangedCB_menu1_Options_FTOutline()
                                ==> void postmesg()
                        void activateCB_menu1_Help_on()
                                void SelectHelpFile()
```

```
void *create_write_BDFfSBD()
        void *_Uxbuild_writeBDFfSBD()
                void cancelCB_writeBDFfSBD()
                void helpCB_writeBDFfSBD()
                        ==> void SelectHelpFile()
                void okCallback_writeBDFfSBD()
                        ==> void write_file()
                        char *extract_string()

void *create_FTADataViewerbBD()
        void *_Uxbuild_FTADataViewerbBD()
                void focusCB_FTADataViewerbBD()
                void createCB_FTADVform()
                void activateCB_FTADVClosepB()
                        void change_seg_attrib()      ***RECURSIVE***
                        void Update_Display()
                                ==> void SetBusyPointer()
                                ==> void ResetView()
                        ==> void SetBusyPointer()

void *create_animationCtlsbBD()
        void *_Uxbuild_animationCtlsbBD()
                void focusCB_animationCtlsbBD()
                void activateCB_playReversedB1()
                        void PlayAnimation()
                                void SelectView()
                                        ==> void Update_Display()
                                void time_style_segment()
                                        void selcontcol()
                                                void selcolor()
                                        ==> void Update_Display()
                                int TestEvent()
                void activateCB_playForwarddB()
                        ==> void PlayAnimation()
                void activateCB_resetAnimdB1()
                        int ResetAnimControls()
                                ==> void Update_Display()
                        ==> void PlayAnimation()
                void activateCB_playStopdB()
                        ==> void Update_Display()
                void activateCB_tCycle_b1()
                void activateCB_tCycle_b2()
                void activateCB_tCycle_b3()
```

```
                void activateCB_stepIncDecaB()
                void activateCB_stepIncIncaB()
                void activateCB_stepInctF()
                        ==> void postmesg()
                void activateCB_currStepDecaB()
                        ==> void PlayAnimation()
                void activateCB_currStepIncaB()
                        ==> void PlayAnimation()
                void activateCB_currSteptF()
                        ==> void PlayAnimation()
                        ==> int ResetAnimControls()
                                ==> void Update_Display()

void *create_confirmWarningwD()   **CURRENTLY UNUSED**

void *create_confirmExitqD()
        void *_Uxbuild_confirmExitqD()
                void cancelCB_confirmExitqD()
                        UxPopdownInterface(confirmExitqD)
                void helpCB_confirmExitqD()
                void okCallback_confirmExitqD()
                        UxPopdownInterface(confirmExitqD)
                        exit ()

void *create_helpWindowbBD()
        void *_Uxbuild_helpWindowbBD()
                void activateCB_pushButton1()
                        UxPopdownInterface(helpWindowbBD)

void *create_selectViewbBD()
        void *_Uxbuild_selectViewbBD()
                void focusCB_selectViewbBD()
                void valueChangedCB_cameraPosX()
                void valueChangedCB_cameraPosY()
                void valueChangedCB_cameraPosZ()
                void valueChangedCB_cameraTarX()
                void valueChangedCB_cameraTarY()
                void valueChangedCB_cameraTarZ()
                void valueChangedCB_viewAngleS()
                void activateCB_selViewOKpB()
                        UxPopdownInterface(selViewOKpB)
                        ==> void SelectView()
                void activateCB_selViewResetpB()
                        ==> int ResetSelectView()
```

```
                void activateCB_selViewApplypB()
                        ==> void SelectView()
                void activateCB_selViewCancelpB()
                        UxPopdownInterface(selViewOKpB)
                        ==> int ResetSelectView()

void *create_DisplayWindow()
        void *_Uxbuild_DisplayWindow()
                void setup_GL()         **SGI CONDITIONAL CODE**
                void test_overlay()     **SGI CONDITIONAL CODE**
                void get_best_visual()
                void createCB_DisplayWindow()
                void exposeCB_DisplayWindow()
                void inputCB_DisplayWindow()
                void resizeCB_DisplayWindow()

void *create_advancedEditorbBD()
        void *_Uxbuild_advancedEditorbBD()
                void focusCB_advancedEditorbBD()
                void valueChangedCB_advEdGrouptF()
                        ==> void postmesg()
                void valueChangedCB_advEdCompFirsttF()
                        ==> void postmesg()
                void losingFocusCB_advEdCompFirsttF()
                void valueChangedCB_advEdCompLasttF()
                        ==> void postmesg()
                void losingFocusCB_advEdCompLasttF()
                        ==> void postmesg()
                void valueChangedCB_rotatetB()
                void valueChangedCB_rotateXtF()
                void valueChangedCB_rotateYtF()
                void valueChangedCB_rotateZtF()
                void valueChangedCB_scaletB()
                void valueChangedCB_scaleXtF()
                        ==> void postmesg()
                void losingFocusCB_scaleXtF()
                        void SetScaleVal()
                void valueChangedCB_scaleYtF()
                        ==> void postmesg()
                void losingFocusCB_scaleYtF()
                        ==> void SetScaleVal()
                void valueChangedCB_scaleZtF()
                        ==> void postmesg()
                void losingFocusCB_scaleZtF()
```

```
        ==> void SetScaleVal()
void valueChangedCB_translatetB()
void valueChangedCB_translateXtF()
void valueChangedCB_translateYtF()
void valueChangedCB_translateZtF()
void valueChangedCB_refPointXtF()
void valueChangedCB_refPointYtF()
void valueChangedCB_refPointZtF()
void valueChangedCB_mirrortB()
void valueChangedCB_yzPlanetB()
void valueChangedCB_zxPlanetB()
void valueChangedCB_xyPlanetB()
void valueChangedCB_copytB()
void valueChangedCB_copyGrouptF()
        ==> void postmesg()
void valueChangedCB_copyFirsttF()
        ==> void postmesg()
void valueChangedCB_copyLasttF()
        ==> void postmesg()
void losingFocusCB_copyLasttF()
        ==> void postmesg()
void activateCB_advancedEditorDeletepB()
        void delete_components()
                ==> void UpdateDisplay()
                ==> void postmesg()
void activateCB_advancedEditorPreviewpB
        void preview_transform()
                void compute_transform()
                ==> void UpdateDisplay()
void activateCB_advancedEditorOKpB()
        UxPopdownInterface(advancedEditorbBD)
        void ApplyTransforms()
                ==> void compute_transform()
                void transform_components()
                        ==> void build_geom()
                        ==> void postmesg()
                void copy_components()
                        ==> void postmesg()
void activateCB_advancedEditorApplypB()
        void ApplyTransforms()
                ==> void compute_transform()
                ==> void transform_components()
                        ==> void build_geom()
                        ==> void postmesg()
```

```
                                    ==> void copy_components()
                                        ==> void postmesg()
                            void activateCB_advancedEditorResetpB()
                                int ResetAdvancedEditor()
                            void activateCB_advancedEditorCancelpB()
                                UxPopdownInterface(advancedEditorbBD)
                                ==> int ResetAdvancedEditor()

void *create_confirmErroreD()
        void *_Uxbuild_confirmErroreD()   **CURRENTLY UNUSED**

void *create_viewDCEDatabBD()
        void *_Uxbuild_viewDCEDatabBD()
                void focusCB_viewDCEDatabBD()
                void activateCB_dvDeleteSelpB()
                void activateCB_dvDeleteAllpB()
                void activateCB_dvUndoDeletepB()
                void activateCB_dvPreviewComppB()
                        void preview_read()
                                int validElemData()
                                int validCompData()
                                int cardContinued()
                                int extract_line()
                        void preview_geom()
                        ==> void postmesg()
                void activateCB_sparepB1()
                void activateCB_sparepB2()
                void activateCB_sparepB3()
                void activateCB_sparepB4()
                void activateCB_sparepB5()
                void activateCB_sparepB6()
                void activateCB_viewDCEDataOKpB()
                        UXpopdownInterface(viewDCEDatabBD)
                        void ResetCardType()
                                void SetCardType()
                                        void UpdateDataViewer()
                                                void GetCardText()
                        int read_bdf()
                                ==> int validElemData()
                                ==> int validCompData()
                                ==> int validHeaderData()
                                ==> int cardContinued()
                                ==> int extract_line()
                                ==> void postmesg()
```

```
                    void build_geom()
                        void build_cylcone()
                            ==> void postmesg()
                        void build_sphere()
                            int get_points()
                                ==> void postmesg()
                        void change_coords()
                        ==> int get_points()
                            ==> void postmesg()
                        void insert_elemlabels()
                        void maxel_err()
                            ==> void postmesg()
                        void maxmin()
                        void seg_setup()
                            ==> void selcontcol()
                                ==> void selcolor()
                        void change_key()
                        ==> void UpdateDisplay()
                            void ResetView()
                            void SetBusyPointer()
                        ==> void ResetSelGroups()
                        ==> void postmesg()
                void activateCB_viewDCEDataApplypB()
                    ==> int read_bdf()                    +++
                    ==> void build_geom()                 +++
                void activateCB_viewDCEDataResetpB()
                void activateCB_viewDCEDataCancelpB()
                    UXpopdownInterface(viewDCEDatabBD)

void *create_fileOpenOpsbBD()
    void *_Uxbuild_fileOpenOpsbBD()
        void valueChangedCB_openFTOpsBDFtB()
        void valueChangedCB_openFTOpsCDFtB()
        void valueChangedCB_openFTOpsGDFtB()
        void valueChangedCB_openFTOpsFDFtB()
        void activateCB_openFTOpsReadpB()
            UXpopdownInterface(fileOpenOpsbBD)
        void activateCB_openFTOpsCancelpB()
            UXpopdownInterface(fileOpenOpsbBD)
            void ResetReadFileOps()
        void valueChangedCB_openFTOpsRAlltB()
            ==> void ResetReadFileOps()
        void valueChangedCB_openFTOpsAddOtB()
            ==> void ResetReadFileOps()
```

```
                    void valueChangedCB_openFTOpsAddNOtB()
                        ==> void ResetReadFileOps()

void *create_orthoSlicerbBD()
        void *_Uxbuild_orthoSlicerbBD()
                    void focusCB_orthoSlicerbBD()
                    void valueChangedCB_positiveXtB()
                    void valueChangedCB_negativeXtB()
                    void valueChangedCB_positiveYtB()
                    void valueChangedCB_negativeYtB()
                    void valueChangedCB_positiveZtB()
                    void valueChangedCB_negativeZtB()
                    void valueChangedCB_positiveXtF()
                    void valueChangedCB_negativeXtF()
                    void valueChangedCB_positiveYtF()
                    void valueChangedCB_negativeYtF()
                    void valueChangedCB_positiveZtF()
                    void valueChangedCB_negativeZtF()
                    void activateCB_ortoSLicerOKpB()
                            UXpopdownInterface(orthoSlicerbBD)
                            void SetCuttingPlanes()
                                ==> void UpdateDisplay()
                    void activateCB_ortoSlicerApplypB()
                            void SetCuttingPlanes()                    +++
                    void activateCB_ortoSlicerResetpB()
                            void ResetOrthoSlicer()
                    void activateCB_ortoSlicerCancelpB()
                            UXpopdownInterface(orthoSlicerbBD)
                            void ResetOrthoSlicer()                    +++

void *create_viewCtlsbBD()
        void *_Uxbuild_viewCtlsbBD()
                    void focusCB_viewCtlsbBD()
                    void activateCB_hSurfs_b1()
                            ==> void UpdateDisplay()
                    void activateCB_hSurfs_b2()
                            ==> void UpdateDisplay()
                    void activateCB_hSurfs_b3()
                            ==> void UpdateDisplay()
                    void valueChangedCB_refAxestB()
                            ==> void UpdateDisplay()
                    void valueChangedCB_labelELEMtB()
                            ==> void UpdateDisplay()
                    void valueChangedCB_labelGRIDtB()
```

```
        ==> void UpdateDisplay()
void valueChangedCB_colorBartB()
        ==> void UpdateDisplay()
void activateCB_zoomOutpB()
        ==> void UpdateDisplay()
void activateCB_zoomInpB()
        ==> void UpdateDisplay()
void activateCB_selectViewpB()
void activateCB_resetViewpB()
        void ResetViewControls()
                void ResetGeomControls()
                ==> void UpdateDisplay()
        void ResetView()
                ==> void UpdateDisplay()
void activateCB_orthoSlicerpB()
void activateCB_geomOpspB()
void activateCB_viewType_Ortho()
        ==> void UpdateDisplay()
void activateCB_viewType_Persp()
        ==> void UpdateDisplay()
void valueChangedCB_azimuthSc()
        ==> void UpdateDisplay()
void valueChangedCB_elevationSc()
        ==> void UpdateDisplay()
void activateCB_presetAZEL_0_0()
        ==> void UpdateDisplay()
void activateCB_presetAZEL_45_0()
        ==> void UpdateDisplay()
void activateCB_presetAZEL_90_0()
        ==> void UpdateDisplay()
void activateCB_presetAZEL_135_0()
        ==> void UpdateDisplay()
void activateCB_presetAZEL_180_0()
        ==> void UpdateDisplay()
void activateCB_presetAZEL_225_0()
        ==> void UpdateDisplay()
void activateCB_presetAZEL_270_0()
        ==> void UpdateDisplay()
void activateCB_presetAZEL_315_0()
        ==> void UpdateDisplay()
void activateCB_presetAZEL_0_45()
        ==> void UpdateDisplay()
void activateCB_presetAZEL_45_45()
        ==> void UpdateDisplay()
```

```
                    void activateCB_presetAZEL_90_45()
                            ==> void UpdateDisplay()
                    void activateCB_presetAZEL_135_45()
                            ==> void UpdateDisplay()
                    void activateCB_presetAZEL_180_45()
                            ==> void UpdateDisplay()
                    void activateCB_presetAZEL_225_45()
                            ==> void UpdateDisplay()
                    void activateCB_presetAZEL_270_45()
                            ==> void UpdateDisplay()
                    void activateCB_presetAZEL_315_45()
                            ==> void UpdateDisplay()
                    void activateCB_presetAZEL_0_n45()
                            ==> void UpdateDisplay()
                    void activateCB_presetAZEL_45_n45()
                            ==> void UpdateDisplay()
                    void activateCB_presetAZEL_90_n45()
                            ==> void UpdateDisplay()
                    void activateCB_presetAZEL_135_n45()
                            ==> void UpdateDisplay()
                    void activateCB_presetAZEL_180_n45()
                            ==> void UpdateDisplay()
                    void activateCB_presetAZEL_225_n45()
                            ==> void UpdateDisplay()
                    void activateCB_presetAZEL_270_n45()
                            ==> void UpdateDisplay()
                    void activateCB_presetAZEL_315_n45()
                            ==> void UpdateDisplay()
                    void activateCB_presetAZEL_0_90()
                            ==> void UpdateDisplay()
                    void activateCB_presetAZEL_0_n90()
                            ==> void UpdateDisplay()

void *create_geomDispOpsbBD()
        void *_Uxbuild_geomDispOpsbBD()
                    void focusCB_geomDispOpsbBD()
                    void activateCB_geomDispOpsOKpB()
                            void SetGeomDispOps()
                                    ==> void UpdateDisplay()
                                    ==> void SetBusyPointer()
                    void activateCB_geomDispOpsApplypB()
                            void SetGeomDispOps()                    +++
                    void activateCB_geomDispOpsResetpB()
                            int ResetGeomDispOps()
```

```
                    void ResetSelGroups()
                    void SetGroupInfo()
                            void setgeomcolwin()
void activateCB_geomDispOpsCancelpB()
        int ResetGeomDispOps()                    +++
void valueChangedCB_groupNametF()
void valueChangedCB_grouptB0()
        void SetGroupInfo()                +++
void valueChangedCB_grouptB1()
        void SetGroupInfo()                +++
void valueChangedCB_grouptB2()
        void SetGroupInfo()                +++
void valueChangedCB_grouptB3()
        void SetGroupInfo()                +++
void valueChangedCB_grouptB4()
        void SetGroupInfo()                +++
void valueChangedCB_grouptB5()
        void SetGroupInfo()                +++
void valueChangedCB_grouptB6()
        void SetGroupInfo()                +++
void valueChangedCB_grouptB7()
        void SetGroupInfo()                +++
void valueChangedCB_grouptB8()
        void SetGroupInfo()                +++
void valueChangedCB_grouptB9()
        void SetGroupInfo()                +++
void valueChangedCB_grouptB10()
        void SetGroupInfo()                +++
void valueChangedCB_grouptB11()
        void SetGroupInfo()                +++
void valueChangedCB_compFirsttF()
        ==> void postmesg()
void valueChangedCB_compLasttF()
        ==> void postmesg()
void losingFocusCB_compLasttF()
        ==> void postmesg()
void exposeCB_geomColorSeldA()
        ==> void UpdateDisplay()
void singleSelectionCB_geomColorSelsL()
        void setgeomcolwin()
void activateCB_gVisibility_b1()
void activateCB_gVisibility_b2()
void activateCB_gVisibility_b3()
void valueChangedCB_transparencySc()
```

```
                    void activateCB_gShading_b1()
                            ==> void UpdateDisplay()
                    void activateCB_gShading_b2()
                            ==> void UpdateDisplay()
                    void activateCB_gShading_b3()
                            ==> void UpdateDisplay()
                    void activateCB_gShading_b4()
                            ==> void UpdateDisplay()

void *create_openBDFfSBD()
        void *_Uxbuild_openBDFfSBD()
                    void cancelCB_openBDFfSBD()
                    void helpCB_openBDFfSBD()
                            void SelectHelpFile()
                    void okCallback_openBDFfSBD()
                            void wininit()
                                    void colorbar()
                                            void selcontcol()
                                                    void selcolor()
                                    void def_pointer()
                            void rdfastgen()
                                    int read_bdf()
                                            int validElemData()
                                            int validCompData()
                                            int validHeaderData()
                                            int cardContinued()
                                            int extract_line()
                                            ==> void postmesg()
                                    void build_geom()
                                            void build_cylcone()
                                            void build_sphere()
                                            void change_coords()
                                            int get_points()
                                            void insert_elemlabels()
                                            void maxel_err()
                                            void maxmin()
                                            void seg_setup()
                                            void change_key()
                                            ==> void UpdateDisplay()
                                            ==> void ResetSelGroups()
                                            ==> void postmesg()
                                    void camcalc()
                                    void init_labels()
                                    void setaxes()
```

```
                                        void UpdateDisplay()
                                                ==> void SetBusyPointer()
                                                ==> void ResetView()
                                        void ResetSelGroups()
                                        void SetBusyPointer()
                                        void SetGeomDispOps()
                                                ==> void UpdateDisplay()
                                                ==> void SetBusyPointer()
                                        void postmesg()
                                        int ResetGeomDispOps()
                                                void SetGroupInfo()
                                                ==> void ResetSelGroups()
                                        ==> void ResetView()

void *create_openCDFfSBD()
        void *_Uxbuild_openCDFfSBD()
                void cancelCB_openCDFfSBD()
                void helpCB_openCDFfSBD()
                        void SelectHelpFile()
                void okCallback_openCDFfSBD()
                        void rdcontour()
                                int validcdkey()
                                ==> void postmesg()
                        ==> char *extract_string()

void *create_openGDFfSBD()
        void *_Uxbuild_openGDFfSBD()
                void cancelCB_openGDFfSBD()
                void helpCB_openGDFfSBD()
                        void SelectHelpFile()
                void okCallback_openGDFfSBD()
                        void rdgrpdef()
                                int validgdkey()
                                ==> void SetBusyPointer()
                                ==> void postmesg()
                        ==> char *extract_string()

void *create_openFDFfSBD()
        void *_Uxbuild_openFDFfSBD()
                void cancelCB_openFDFfSBD()
                void helpCB_openFDFfSBD()
                void okCallback_openFDFfSBD()
                        void rdFTA()
                                int validftakey()
```

```
                                    ==> void postmesg()
                                    void AddFTADVEntry()
                                        void activateCBFTADVentries()
                                            void change_seg_attrib()
                                            ==> void UpdateDisplay()
                                            ==> void SetBusyPointer()
                                ==> char *extract_string()

void *create_bdfCardsbBD()
        void *_Uxbuild_bdfCardsbBD()
                void focusCB_bdfCardsbBD()
                void valueChangedCB_textField1()
                void losingFocusCB_textField1()
                void valueChangedCB_textField2()
                void losingFocusCB_textField2()
                        void UpdateDataViewer()
                                void GetCardText()
                void valueChangedCB_textField3()
                void losingFocusCB_textField3()
                        void UpdateDataViewer()
                                void GetCardText()
                void valueChangedCB_textField4()
                void losingFocusCB_textField4()
                        void UpdateDataViewer()
                                void GetCardText()
                void valueChangedCB_textField5()
                void losingFocusCB_textField5()
                        void UpdateDataViewer()
                                void GetCardText()
                void valueChangedCB_textField6()
                void losingFocusCB_textField6()
                        void UpdateDataViewer()
                                void GetCardText()
                void valueChangedCB_textField7()
                void losingFocusCB_textField7()
                        void UpdateDataViewer()
                                void GetCardText()
                void valueChangedCB_textField8()
                void losingFocusCB_textField8()
                        void UpdateDataViewer()
                                void GetCardText()
                void valueChangedCB_textField9()
                void losingFocusCB_textField9()
                        void UpdateDataViewer()
```

```
                    void GetCardText()
void valueChangedCB_textField10()
void losingFocusCB_textField10()
          void UpdateDataViewer()
                    void GetCardText()
void valueChangedCB_textField11()
void losingFocusCB_textField11()
          void UpdateDataViewer()
                    void GetCardText()
void valueChangedCB_textField12()
void losingFocusCB_textField12()
          void UpdateDataViewer()
                    void GetCardText()
void valueChangedCB_textField13()
void losingFocusCB_textField13()
          void UpdateDataViewer()
                    void GetCardText()
void valueChangedCB_textField14()
void losingFocusCB_textField14()
          void UpdateDataViewer()
                    void GetCardText()
void valueChangedCB_textField15()
void losingFocusCB_textField15()
          void UpdateDataViewer()
                    void GetCardText()
void valueChangedCB_textField16()
void losingFocusCB_textField16()
          void UpdateDataViewer()
                    void GetCardText()
void valueChangedCB_textField17()
void losingFocusCB_textField17()
          void UpdateDataViewer()
                    void GetCardText()
void valueChangedCB_textField18()
void losingFocusCB_textField18()
          void UpdateDataViewer()
                    void GetCardText()
void valueChangedCB_textField19()
void losingFocusCB_textField19()
          void UpdateDataViewer()
                    void GetCardText()
void valueChangedCB_textField20()
void losingFocusCB_textField20()
          void UpdateDataViewer()
```

```
                void GetCardText()
void activateCB_bdfCardsApplypB()
        void UpdateDataViewer()
                void GetCardText()
void activateCB_bdfCardsResetpB()
        void SetCardType()
                void UpdateDataViewer()
                        void GetCardText()
void activateCB_bdfCards_ccone1()
        void SetCardType()          +++
void activateCB_bdfCards_ccone2()
        void SetCardType()          +++
void activateCB_bdfCards_chex1()
        void SetCardType()          +++
void activateCB_bdfCards_chex2()
        void SetCardType()          +++
void activateCB_bdfCards_cline()
        void SetCardType()          +++
void activateCB_bdfCards_cquad()
        void SetCardType()          +++
void activateCB_bdfCards_csphere()
        void SetCardType()          +++
void activateCB_bdfCards_ctri()
        void SetCardType()          +++
void activateCB_bdfCards_grid()
        void SetCardType()          +++
void activateCB_bdfCards_section()
        void SetCardType()          +++
void activateCB_bdfCards_vehicle()
        void SetCardType()          +++
void activateCB_bdfCards_wall()
        void SetCardType()          +++
void activateCB_bdfCards_comment()
        void SetCardType()          +++
void activateCB_bdfCards_name()
        void SetCardType()          +++
void activateCB_bdfCards_cordae()
        void SetCardType()          +++
void activateCB_bdfCards_point()
        void SetCardType()          +++
void activateCB_bdfCards_unitgbl()
        void SetCardType()          +++
void activateCB_bdfCards_unitlcl()
        void SetCardType()          +++
```

```
                    void activateCB_cdfCards_comment()
                            void SetCardType()              +++
                    void activateCB_gdfCards_comment()
                            void SetCardType()              +++
                    void activateCB_caseCards_comment()
                            void SetCardType()              +++
                    void valueChangedCB_dceGrouptF()
                            ==> void postmesg()
                    void activateCB_dceComptF()
                            void ResetCardType()            +++
                            void SetDataViewer()
                    void valueChangedCB_dceComptF()
                            ==> void postmesg()
                    void valueChangedCB_dceEditNewtB()
                            void ResetCardType()            +++
                            void ResetDataViewer()
                    void valueChangedCB_dceEditExistingtB()
                            void ResetCardType()            +++
                            void ResetDataViewer()
                    void valueChangedCB_dceEditHeadertB()
                            void ResetCardType()            +++
                            void SetDataViewer()

void *create_printOpsbBD()
        void *_Uxbuild_printOpsbBD()
                    void focusCB_printOpsbBD()
                    void valueChangedCB_printPStB()
                            int SetPOSize()
                    void valueChangedCB_printEPStB()
                            int SetPOSize()
                    void valueChangedCB_printHPGLtB()
                            int SetPOSize()
                    void valueChangedCB_printCGMtB()
                            int SetPOSize()
                    void valueChangedCB_printPICTtB()
                            int SetPOSize()
                    void valueChangedCB_printAtB()
                    void valueChangedCB_printBtB()
                    void valueChangedCB_printCtB()
                    void valueChangedCB_printDtB()
                    void valueChangedCB_printEtB()
                    void valueChangedCB_printLandtB()
                    void valueChangedCB_printPorttB()
                    void valueChangedCB_printPrintertB()
```

```
                    void valueChangedCB_printFiletB()
                    void valueChangedCB_printFiletF()
                    void valueChangedCB_printPrintertF()
                    void valueChangedCB_printNCopytF()
                    void activateCB_printOKpB1()
                            void PrintHardcopy()
                                    ==> void SetBusyPointer()
                                    ==> void postmesg()
                    void activateCB_printResetpB()
                            int ResetPrintOps()
                                    ==> void SetBusyPointer()
                                    ==> void postmesg()
                                    void SetPOFileType()
                                    void SetPOSize()
                                    void SetPOOrientation()
                                    void SetPOOutput()
                    void activateCB_printCancelpB()

void *create_confirmFCLosswD()
        void *_Uxbuild_confirmFCLosswD()
                void okCallback_confirmFCLosswD()
```

Appendix C: Interface Motif Widget Summary

The following is list of the primary widgets used in the VISAGE GUI. Each widget's function, relevant attributes, and utility functions are listed.

XmArrowButton

Function: an arrow-shaped push-button

Attributes: XmNactivateCallback - list of callbacks for button activation

Utilities: none

XmBulletinBoard

Function: provides simple geometry management for children widgets base for all popup dialogs (only bulletin board dialogs in VISAGE)

Attributes: XmNdialogTitle - sets title of dialog

Utilities: none

XmDialogShell

Function: used for message dialogs

Attributes: XmNdialogType - specifies dialog type
XmDIALOG_ERROR
XmDIALOG_QUESTION
XmDIALOG_WARNING
XmNcancelCallback - list of callbacks for cancel
XmNhelpCallback - list of callbacks for help
XmNokCallback - list of callbacks for ok

Utilities: none

XmFileSelectionBox

Function: used to select files

Attributes:
- XmNdialogType - specifies dialog type
  XmDIALOG_FILE_SELECTION
- XmNpattern - specifies file search pattern
- XmNcancelCallback - list of callbacks for cancel
- XmNhelpCallback - list of callbacks for help
- XmNokCallback - list of callbacks for ok

Utilities: none

XmLabel

Function: used for labeling of widgets

Attributes: XmNlabelString - text string for label

Utilities: none

XmList

Function: used to select 1 or more items from a group of choices
all VISAGE lists are Single Select

Attributes:
- XmNitemCount - specifies total number of items
- XmNitems - points to array of compond string list items
- XmNselectionPolicy - defines selection action
  XmSINGLE_SELECT
- XmNsingleSelectionCallback - list of callbacks for single selection
- XmNvisibleItemCount - specifies number of items visible

Utilities:
- XmListAddItem - adds item to list
- XmListDeselectAllItems - deselect all items in list
- XmListSelectPos - select a position in a list
- XmListItemPos - returns item position

XmPushButton

Function: issues commands

Attributes:
- XmNlabelString - text string for label
- XmNactivateCallback - list of callbacks for activation

Utilities: none

XmRowColumn

Function: used for row/column ordering, radio boxes, menus, menu bars

Attributes: XmNrowColumnType - set type of row/column
XmWORK_AREA (default)
XmMENU_BAR
XmMENU_POPUP
XmMENU_PULLDOWN
XmMENU_OPTION

for menus:
XmCascadeButton - used in menus
XmNmenuHistory - used to set options menus

for radio buttons:
XmNradioBehavior - enforces behavior on toggle buttons

XmScale

Function: value selector

Attributes: XmNdecimalPoints - to specify decimal places in value
XmNmaximum - specify max value (> min value)
XmNminimum - specify min value (< max value)
XmNscaleMultiple - specifies amount to move slider
XmNshowValue - specify whether to show current value
XmNvalue - current slider position (min <= x <= max)
XmNtitleString - text string for title
XmNvalueChangedCallback - list of callbacks for value change

Utilities: XmScaleGetValue - gets scale value
XmScaleSetValue - sets scale value

XmScrolledWindow

Function: combines 1 or 2 ScrollBars and a viewing area for window onto larger data display default values used

Attributes: none relevant

Utilities: none

XmText

Function: provides single-line and multi-line text editor

Attributes:
- XmNeditable - specifies whether text is editable
- XmNeditMode - specifies set of keyboard bindings
  - XmMULTI_LINE_EDIT
- XmNvalue - specifies string value (char *)
- XmNcolumns - specifies initial width in characters
- XmNrows - specifies initial height in characters
- XmNactivateCallback - list of callbacks for activation
- XmNvalueChangedCallback - list of callbacks for text changes

Utilities:
- XmTextGetString - get text string value
- XmTextSetString - set text string value
- XmTextGetInsertionPosition - get insertion position
- XmTextSetInsertionPosition - set insertion position
- XmTextInsert - insert text in text string
- XmTextReplace - replace part of text string
- XmTextRemove - remove selected text

XmTextField

Function: single line text editor

Attributes:
- XmNeditable - specifies whether text field is editable
- XmNvalue - specifies string value (char *)
- XmNactivateCallback - list of callbacks for carriage return
- XmNvalueChangedCallback - list of callbacks when text is typed

Utilities:
- XmTextFieldGetValue - gets text value
- XmTextFieldSetValue - sets text value

XmToggleButton

Function: set non-transitory state data

Attributes:
- XmNindicatorType - set type of toggle button
  - XmONE_OF_MANY = radio button
  - XmN_OF_MANY = toggle button
- XmNlabelString - text string for label
- XmNvalueChangedCallback - list of callbacks for value change

Utilities:
- XmToggleButtonGetState - get toggle button state
- XmToggleButtonSetState - set toggle button state

Appendix D: VISAGE 2.2 User's Manual

The VISAGE 2.2 User's Manual is found on the following pages.

VISAGE 2.2 User's Manual

Brett F. Grimes, 88 CG/SCSA

December 1994

A program for displaying and editing FASTGEN 4 data sets

SCSA Technical Report
Project SCSA-120

Table of Contents

1.0 Introduction 101
- 1.1 Purpose 101
- 1.2 New Features 101
  - 1.2.1 Advanced Rendering 101
  - 1.2.2 Target Description Editing 101
  - 1.2.3 Fault Tree Data Representation 101
- 1.3 Planned Enhancements for VISAGE 101
  - 1.3.1 Line-of-Sight (LOS) Data Representation 101
  - 1.3.2 Target Description Animation 101
- 1.4 GUI Description 102
  - 1.4.1 Overview 102
  - 1.4.2 Open Software Foundation (OSF) Motif Widgets 104
  - 1.4.3 Widget Colors 106
- 1.5 Installation 113
- 1.6 Operating System Requirements 113
- 1.7 Memory Requirements 113
- 1.8 Feedback 114
- 1.9 Other Issues 114
  - 1.9.1 Sun Version 114
  - 1.9.2 SGI Version 114
  - 1.9.3 All Versions 114

2.0 Using VISAGE 115
- 2.1 The Menu Bar 115
  - 2.1.1 File Menu 115
  - 2.1.2 Edit Menu 115
  - 2.1.3 Options Menu 116
  - 2.1.4 Help Menu 116
- 2.2 Open File Types Reference 117
- 2.3 File Selectors 118
- 2.4 View Controls Reference 119
- 2.5 Geometry Controls Reference 122
- 2.6 Animation Controls Reference 124
- 2.7 Orthogonal Slicer Reference 125
- 2.8 Select View Reference 126
- 2.9 Basic Editor Reference 127
- 2.10 Advanced Editor Reference 130
- 2.11 Fault Tree Data Viewer Reference 133
- 2.12 Mouse Functions 134
  - 2.12.1 Select Geometry 134

2.12.2 Select Camera Target 134
2.12.3 Quick Zoom 134

3.0 Data File Formats 135
3.1 Introduction 135
3.2 VISAGE Data Deck 135
CORDAE 137
POINT 138
UNITGBL 139
UNITLCL 140
3.3 Contour Data File (CDF) 141
CD 142
LIMITS 143
TMSTEPS 144
VIEW 145
3.4 Group Defaults File (GDF) 146
COLORGR 147
GRPNAME 149
3.5 Fault Tree Data File (FDF) 150
FTSET 152
FTFUNC 153
FTSYS 154
FTSUB 155
FTCOMP 156
$CNAME 157

Appendix A: Bulk Data File Example 159
Appendix B: Contour Data File Example 167
Appendix C: Group Defaults File Example 168
Appendix D: Fault Tree Data File Example 169
Appendix E: Fault Tree Output File Example 171

List of Figures

FIGURE 1 VISAGE (F-15 with Cutting Plane through Port-side Engine) 107
FIGURE 2 Shaded and Transparent Image of an F-15 Target Description 108
FIGURE 3 VISAGE Data Card Editor and Data Viewer 109
FIGURE 4 VISAGE Advanced Editor 110
FIGURE 5 VISAGE Fault Tree Data Viewer 111
FIGURE 6 VISAGE Popup Dialogs 112
FIGURE 7 File Selectors 118
FIGURE 8 VISAGE Data Deck Order 136
FIGURE 9 CDF Data Deck Order 141
FIGURE 10 GDF Data Deck Order 146
FIGURE 11 FDF Data Deck Order 151

List of Tables

TABLE 1 Open File Types Reference 117
TABLE 2 View Controls Reference 119
TABLE 3 Geometry Controls Reference 122
TABLE 4 Animation Controls Reference 124
TABLE 5 Orthogonal Slicer Reference 125
TABLE 6 Select View Reference 126
TABLE 7 Data Card Editor Reference 127
TABLE 8 Data Viewer Reference 129
TABLE 9 Advanced Editor Reference 131
TABLE 10 Fault Tree Data Viewer Reference 133

1.0 Introduction

1.1 Purpose

VISAGE is a Graphical User Interface (GUI) designed for the visualization and interrogation of FASTGEN 4 bulk data sets. It is assumed that the user is familiar with FASTGEN 4 formats and technical terms. **VISAGE 2.2 replaces VISAGE 2.1.**

1.2 New Features

1.2.1 Advanced Rendering

VISAGE 2.2 has the capability to produce shaded images with transparency. The flat, Gouraud, and Phong shading algorithms are available.

1.2.2 Target Description Editing

Using the Basic Editor (Data Card Editor and Data Viewer dialogs), the user can now edit and save target descriptions. See Section 2.10 for details on the Basic Editor. Also available is an Advanced Editor that allows the user to rotate, scale, translate, mirror, and copy components. See Section 2.11 for details on the Advanced Editor.

1.2.3 Fault Tree Data Representation

Users are able to visualize fault tree data using a file generated by COVART 4.

1.3 Planned Enhancements for VISAGE

1.3.1 Line-of-Sight (LOS) Data Representation

Users will be able to visualize the LOS results using a data file generated by FASTGEN 4.

1.3.2 Target Description Animation

Expansions to the Contour Data File (CDF) format will allow for the animation of up to 12 separate "objects" (one per Group) as well as the current camera translation features.

1.4 GUI Description

The GUI is based on the OSF Motif Widget Set and is composed of several display and dialog windows:

1.4.1 Overview

1] VISAGE (Main Window)

This window contains a menu bar and a scrolled text area. The menu bar contains the overall application controls. The scrolled text area provides the user with various information relating to the target description and the menu selections.

2] Display Window

This window is where the target description is displayed.

3] Open File Types

Dialog for selecting the types of files to load into VISAGE.

4] Bulk Data File Selector

File selector dialog for loading a FASTGEN Bulk Data File (BDF) into VISAGE.

5] Contour Data File Selector

File selector dialog for loading a VISAGE Contour Data File (CDF) into VISAGE.

6] Fault Tree Data File Selector

File selector dialog for loading a VISAGE Fault Tree Data File (FDF) into VISAGE.

7] Group Defaults File Selector

File selector dialog for loading a VISAGE Group Defaults File (GDF) into VISAGE.

8] Basic Editor

Consists of two dialogs; the Data Card Editor and the Data Viewer. Allows for component-level editing of the target description.

9] Advanced Editor
Allows for copying and spatial manipulation of components, i.e., rotation, scaling, translation, and mirroring.

10] Fault Tree Data Viewer
Allows user to see relationships between COVART 4 fault tree data and the FASTGEN 4 target description.

11] Animation Controls
Controls for data input through the CDF.

12] View Controls
Basic controls governing the viewing aspect of the current target description.

13] Geometry Options
Option settings for the appearance of the current target description.

14] Orthogonal Slicer
Controls for selecting the position and orientation of the orthogonal cutting planes.

15] Select View
Controls for selecting a specific camera view of the current target description.

16] Print Options
Options for hardcopy output of the current view of the target description.

17] Confirm Exit
This is a message dialog asking the user to confirm his decision to exit VISAGE.

18] Help Window
A scrolled text window for the display of help files on assorted VISAGE topics.

19] Color Bar with Index and User-Assigned Values
This window provides the user with a visual of the color bar and any assigned ranges of values the user may have specified for the colors.

1.4.2 Open Software Foundation (OSF) Motif Widgets

The GUI for VISAGE is based on the standard OSF Motif widget set. Motif is quickly becoming the GUI of choice on Unix workstations.

The following is a list of the Motif widgets in the interface and how they are used:

Note: Unless specified, all mouse actions are done using the left mouse button.

1] Push Buttons

Push buttons are widgets that initiate an action or function when clicked on ("pushed") with the mouse.

2] Toggle Buttons

These buttons are widgets that essentially act like "check boxes" and may initiate an action or set an option.

3] Radio Buttons

These buttons are really just toggle buttons, but only one out of a group may be selected at one time. The name is derived from the push buttons for selecting preset stations on radios.

4] Pulldown Menus

This is a menu that the user activates by clicking once on or by holding down the mouse button on the selected menu text. This brings up a "menu" of push buttons for the user to select from.

5] Option Menus

Option menus are activated by holding down the mouse button over the menu and highlighting a selection on the "menu". This selection becomes the "top" of the "menu" and is displayed on the widget.

6] Scales

Scale is the Motif name for a sliding bar. This widget lets the user select a number from a preset range of numbers. Users can elect to "drag" the sliding bar to a selected point using the left mouse button, jump to a selected point using the middle mouse button, or increment or decrement the current selected value by one by clicking to the right or left, respectively, of the sliding bar.

7] Text Fields
These widgets allow the user to type in text.

8] Scrolled Windows
Scrolled windows allow the user to scroll around a window containing more items than can be displayed at once in the allotted window size.

9] Scrolled Lists
Scrolled lists allow the user to scroll through a list of choices and select one item.

10] Scrolled Text
Scrolled text is used in the VISAGE and Help Windows to allow the user to scroll through information that may be larger than the available display area.

11] Message Dialogs
Message dialogs appear to confirm a choice, such as exiting the program.

12] File Selectors
File selectors are compound widgets (composed of several different widgets, but treated as one large one) that allow the user to select files for input or output purposes.

13] Popup Dialogs
These are compound widgets that are created by the GUI designer. These dialogs are used to separate different functions of the program, to compartmentalize the operation and to prevent information overload.

14] Arrow Buttons w/Text Fields
These are compound widgets that are similar in function to the Scales. A user can increment or decrement the value displayed in the associated text field by clicking on either the up or down arrows, respectively. The user can also input a specific value by typing in the text field.

1.4.3 Widget Colors

Color is used to indicate the function and/or the state of a widget. Except as designated below, **Active** widgets will be **Blue** or **Royal Blue** and **Inactive** widgets (separators and labels) will be either **Dark Grey** or **Navy Blue**.

The push buttons are one of two types:

Immediate

Extended

Immediate push buttons are **Blue** in color and will flash red when activated.

Extended push buttons are **Gold** in color and will flash red when "pressed". These push buttons will cause popup dialogs to appear.

Toggle and **Radio** buttons are **Royal Blue** in color and will turn **Red** when toggled to active mode. Some buttons are active by default and are initially **Red**.

Text Fields, **Scrolled Lists**, and **Scrolled Text** are **Grey** with **White** text. They will not change color, except Scrolled Lists, which will reverse highlight the selection.

Scales and **Option Menus** are **Royal Blue** and will not change color.

Indicator Text Fields are **Black** with **Green** text. These cannot be edited and will not change color.

The main **Pulldown Menu Bar** is **Navy Blue** with **Royal Blue** "menus".

Non-button areas and indicators are **Dark Grey**.

Figures 1-6 illustrate features of the VISAGE GUI.

FIGURE 1 VISAGE (F-15 with Cutting Plane through Port-side Engine)

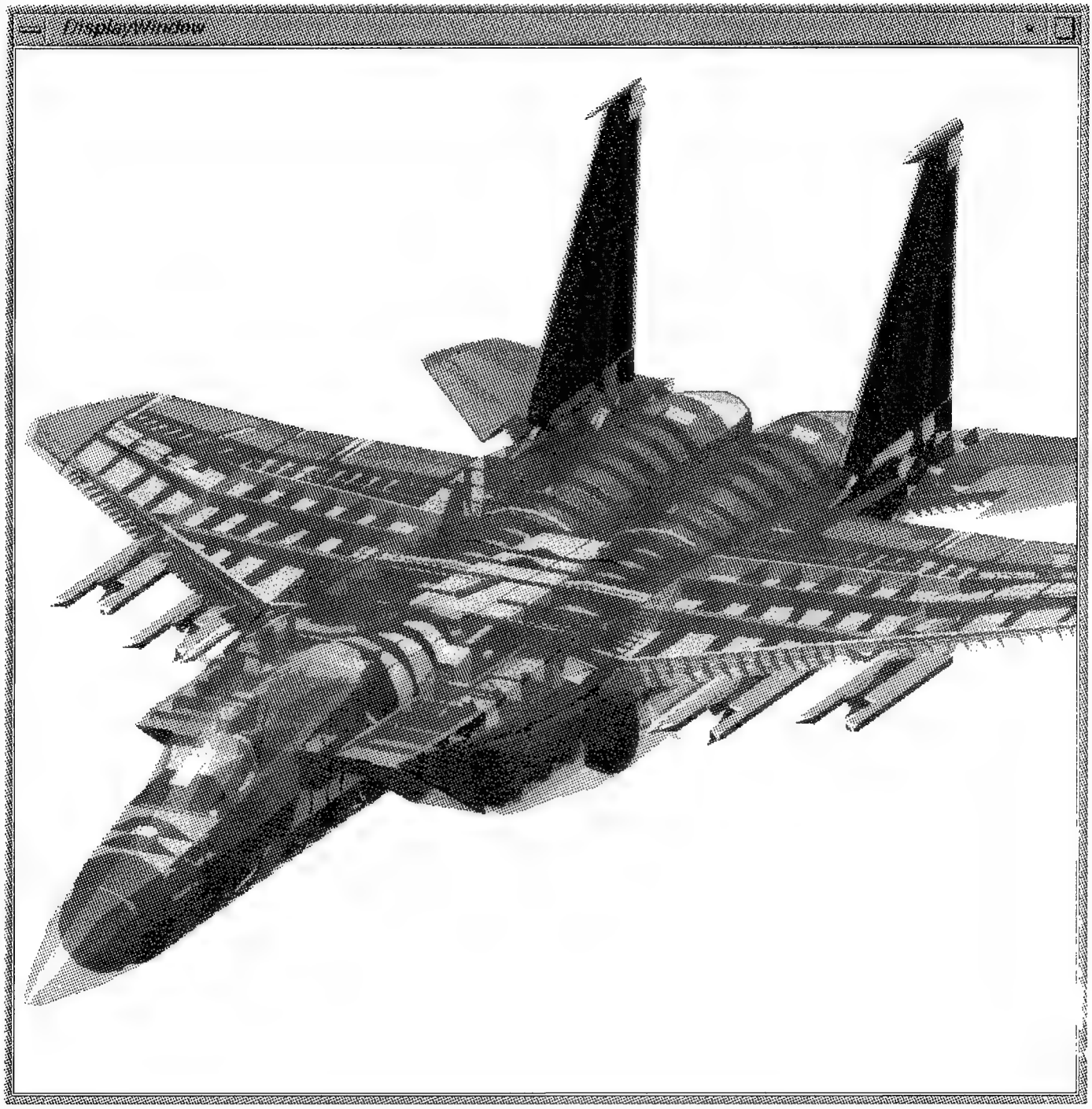


FIGURE 2 Shaded and Transparent Image of an F-15 Target Description

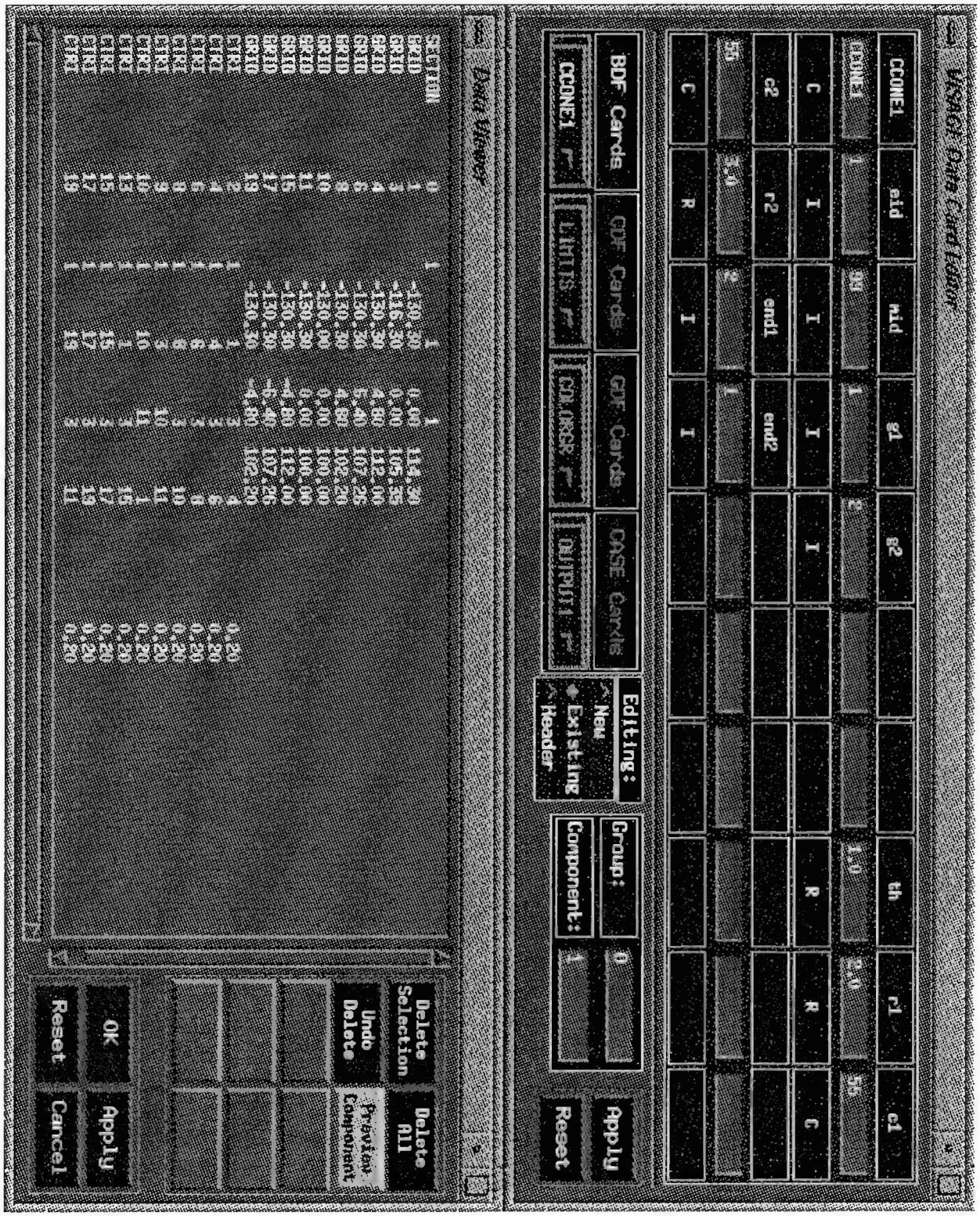


FIGURE 3 VISAGE Data Card Editor and Data Viewer

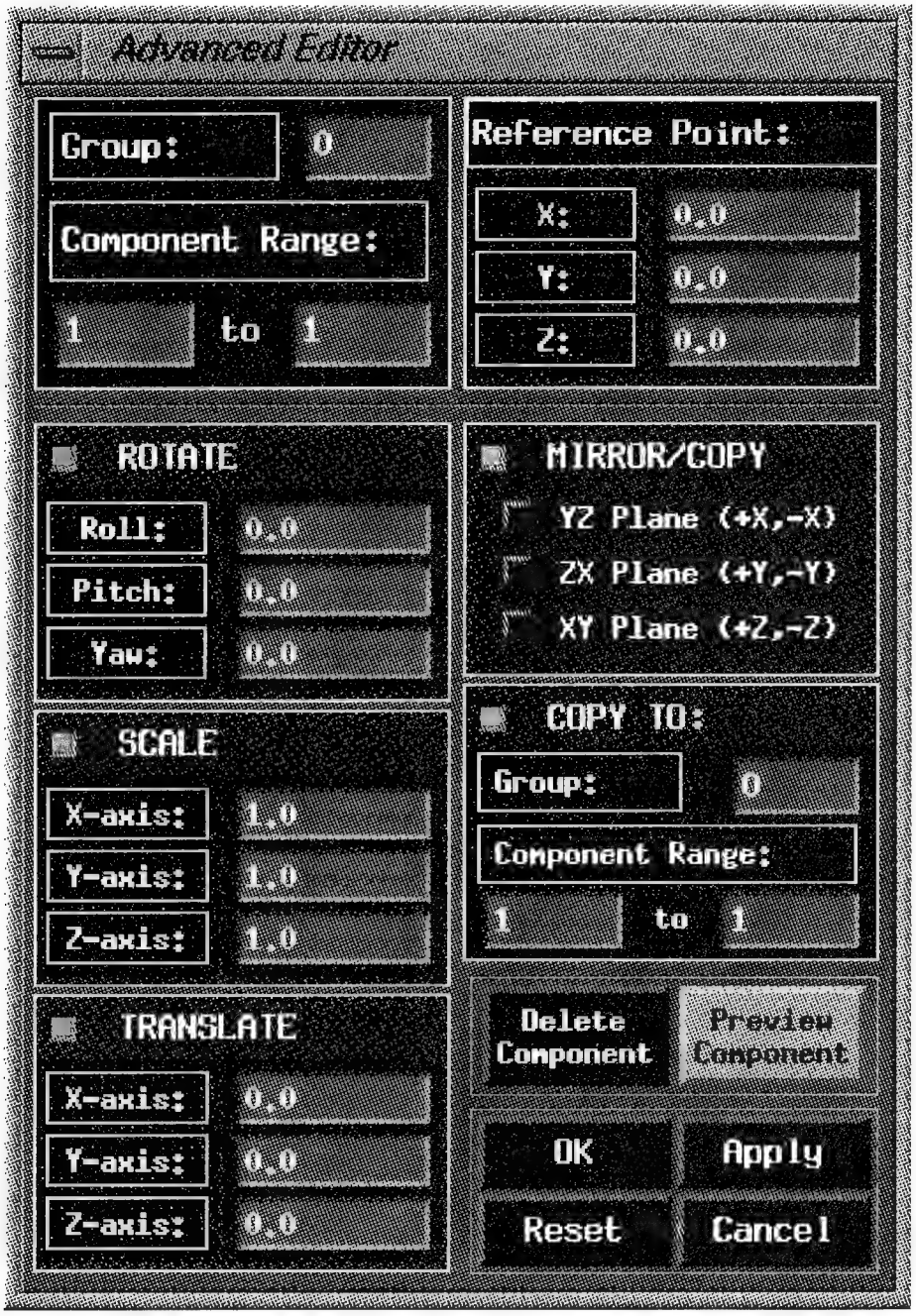


FIGURE 4 VISAGE Advanced Editor

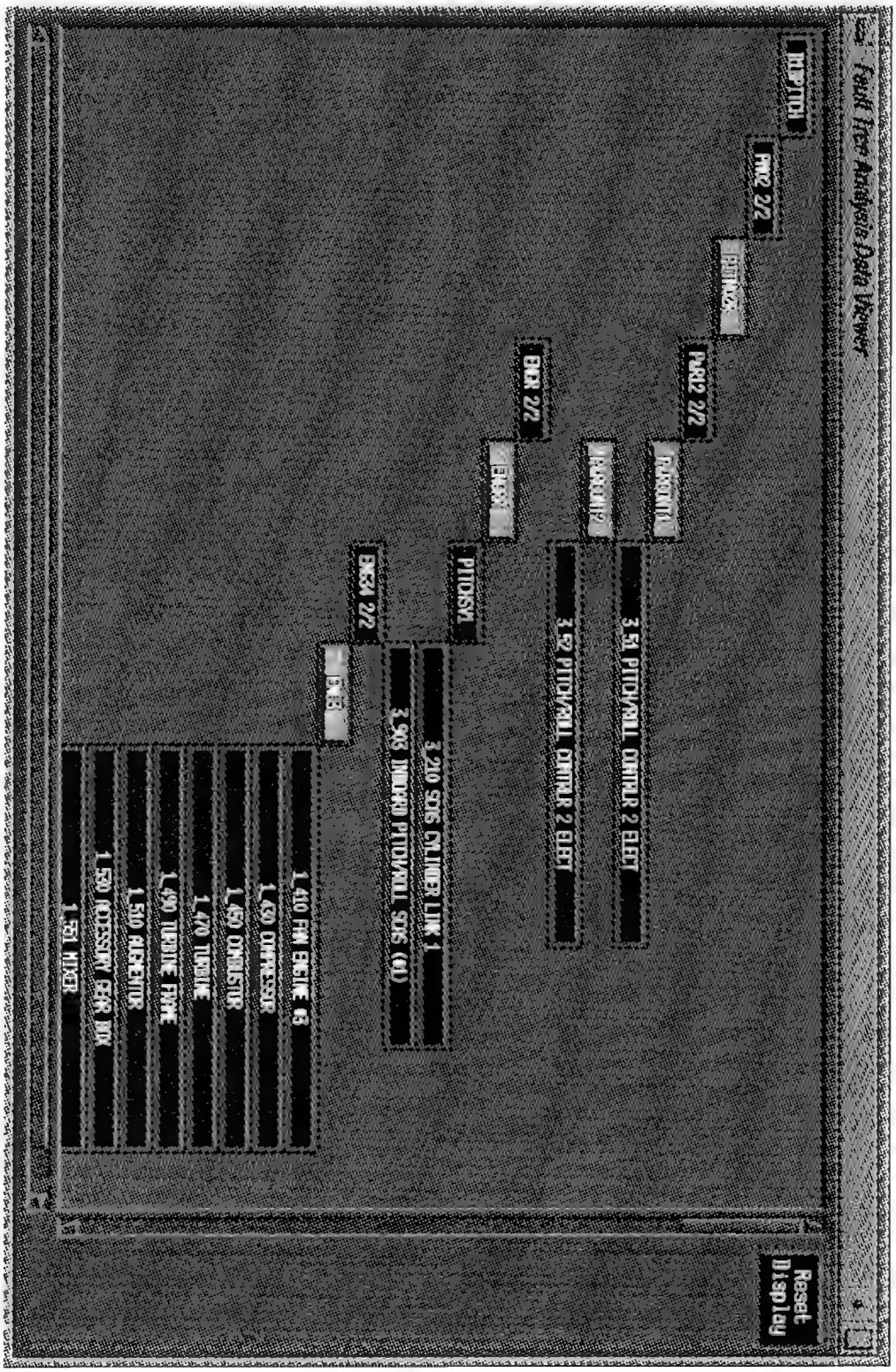


FIGURE 5 VISAGE Fault Tree Data Viewer

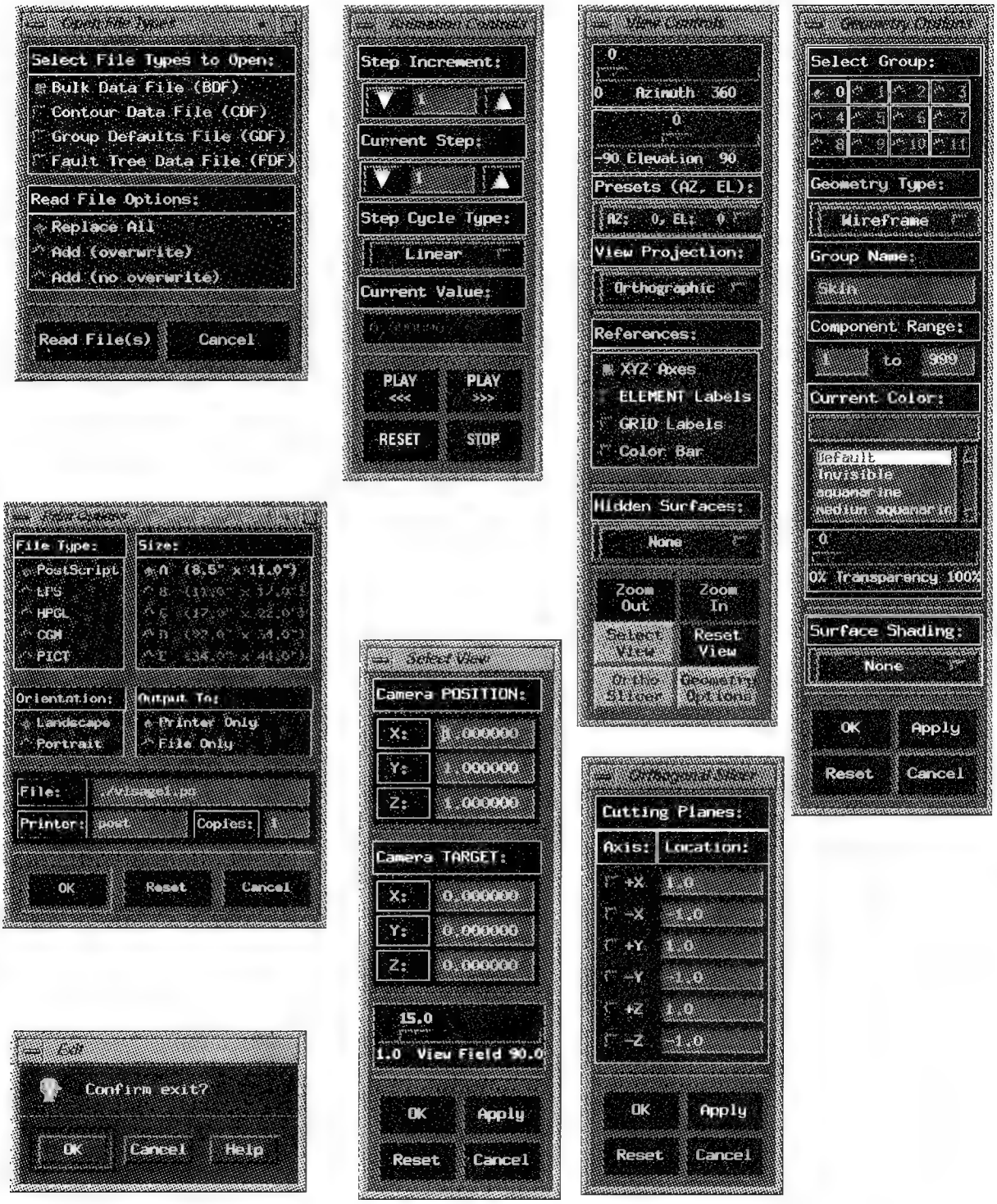


FIGURE 6 VISAGE Popup Dialogs

1.5 Installation

VISAGE is available on the following platforms:

Sun 4's / SPARCstations

Silicon Graphics Inc. Iris 4D, Indigo, Power, Crimson, and Reality Engine series

(DEC available by request)

To install VISAGE simply download all the files on the distribution tape for your particular platform onto your system. The commands to do this are:

(SGI, DEC)	(Sun)	(Solaris)
tar xvp	tar xvpf /dev/rst8	tar xvpf /dev/rmt/0
cd visage_2.2	cd visage_2.2	cd visage_2.2
./install-visage	./install-visage	./install-visage

The files will include the VISAGE executables, a shell script that acts as a front end to the executable, sample data files, an installation script, a de-installation script, and a PostScript copy of this manual. To begin using VISAGE, type in the following:

visage

Optionally, you can direct VISAGE to look for files in a directory other than the one you are executing in by setting the VIS_HOME environment variable, e.g.:

setenv VIS_HOME <directory-pathname>

1.6 Operating System Requirements

Sun: SunOS 4.1.3 (X11R5) or Solaris 2.3 (SunOS 5.3) and above

SGI: IRIX 5.2 and above

1.7 Memory Requirements

The VISAGE program is currently set to allow the user to view files of any size that can be accommodated by the main memory in the user's system, dependent upon the restrictions found in the FASTGEN 4 User's Manual regarding the FASTGEN 4 data files. VISAGE will work best on systems equipped with at least 32 MB of main memory (RAM). Systems with less than this can execute VISAGE, but expect excessive paging of memory on large data files.

1.8 Feedback

Any feedback from users on the interface design and the improved capabilities would be welcome. Please feel free to e-mail any comments and/or problems to me at:

tacit@msrc.wpafb.af.mil

1.9 Other Issues

1.9.1 Sun Version

1] The color bar window cannot be dismissed once you have activated it. This is because the window is created by the HOOPS graphics library and in the current version it does not respond well to X11 and Motif window directions.

2] Related to the color bar problem is the main Display Window. Too much moving or resizing of the Display Window will cause HOOPS to "lose" track of where in the Motif display area the HOOPS window should be located. According to HOOPS, this is a problem only when you display Motif windows under Sun's OpenWindows interface. This will not be an issue if you are running Motif on your Sun systems.

1.9.2 SGI Version

1] In the SGI version, to take advantage of the GL driver, a "mixed-mode" version of a Motif drawing area widget is used. The Display Window will come up in technicolor the first time VISAGE is started up. This has to do with colormap usage and does not cause any problems.

1.9.3 All Versions

1] The ray-tracing capabilities and the problems with its integration in the HOOPS library by Ithaca Software have rendered the ray-tracing option unusable. It will probably not return. The good news is that transparency is now available without the advanced rendering module, so the only capability missing is shadow generation.

2.0 Using VISAGE

2.1 The Menu Bar

The menu bar is the first level of the GUI the user will encounter. The menu bar consists of the following main categories:

File

Edit

Options

Help

2.1.1 File Menu

The File menu selection currently has 5 selections:

1] Open...

This selection will popup the Open File Types dialog.

2] Save

This will save the current target description using the current filename.

3] Save As...

This will bring up the Save As file selector so the user can select a new filename to use for the current target description.

4] Print...

This selection will popup the Print Options dialog.

5] Exit

This selection will popup the Confirm Exit message dialog.

2.1.2 Edit Menu

The Edit menu selection has two selections:

1] Basic Editor

This selection will popup the Data Card Editor and the Data Viewer dialogs.

2] Advanced Editor

This selection will popup the Advanced Editor.

2.1.3 Options Menu

The Options menu has four selections:

1] Allow Independent Scaling

This toggle button allows the Scale function of the Advanced Editor to work independently on each of the X, Y, and Z axes. The default setting is to lock scaling to the same value for all axes.

2] Output Geometry Selection

This toggle button enables the output of data from the mouse button selection of geometry elements. Information on geometry selections is output to a text file that is named by appending the '.sel' suffix to the name of the current Bulk Data File (BDF). The default setting is to not output the geometry selection information.

3] Show Fault Tree Components

This toggle button allows user to set the Fault Tree Data Viewer to display or not display the bottom level components defined in a fault tree. This is useful for displaying large trees when the lowest level components do not need to be displayed. The default setting is to display fault tree components.

4] Output Fault Tree Outline

This toggle button allows user to set the Fault Tree Data Viewer to output a text listing of the fault tree data, whenever a Fault Tree Data File (FDF) is input. This output capability mimics the view of the fault tree in the Fault Tree Data Viewer in a text format. The data is output to a text file that is named by appending the '.txt' suffix to the name of the current FDF. The default setting is to output the fault tree information.

2.1.4 Help Menu

The Help menu selection currently has only 1 selection:

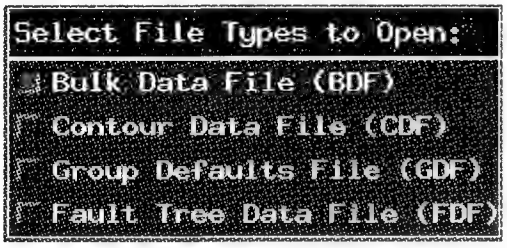
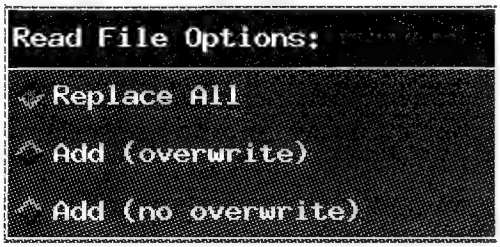
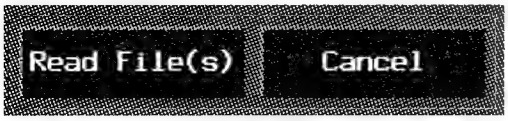
1] On VISAGE

This selection will popup up the Help Window with a short summary of the VISAGE software.

2.2 Open File Types Reference

The Open File Types popup dialog allows the user to load any combination of the 4 types of files expected by VISAGE (see Ch. 3.0); the Contour Data File (CDF), the Group Defaults File (GDF), the Fault Tree Data File (FDF), or the Bulk Data File (BDF). The CDF, GDF, and FDF are optional files and are used in conjunction with the BDFs. The user may visually combine two or more target descriptions by loading the files separately and selecting the appropriate Read File Option. To load in any combination of the three file types (a BDF file must be used if no previous target description has been loaded), the user simply selects the file types and then selects the appropriate files from the File Selectors. These selections are described in Table 1.

TABLE 1 **Open File Types Reference**

Item	Effect
Select File Types to Open: 	These toggle buttons allow the user to select a combination of the four FASTGEN and VISAGE file types; BDF, CDF, GDF, and FDF. The default selection is *Bulk Data File*.
Read File Options: 	These radio buttons allow the user to select the way the new data will overwrite the existing target description data. Choices are to **Replace All**, **Add (overwrite)**, and **Add (no overwrite)**. For BDF's, **Replace All** will completely replace the existing target description. **Add (overwrite)** will replace any existing components found in the new BDF. **Add (no overwrite)** will not replace any existing components found in the new BDF. CDF, GDF, and FDF data will always be completely replaced. The default setting is *Replace All*.
Read File(s)/Cancel 	These push-buttons control the use of the Open File Types dialog. *OK* will bring up the file selectors for the chosen file types and dismiss the popup dialog. *Cancel* will reset the options and dismiss the popup dialog.

2.3 File Selectors

The File Selectors allow the user to select the appropriate files as specified in the Open File Types popup dialog. Each data file type has its own File Selector. File Selectors are brought up in their predefined loading order. Figure 3 below shows the four File Selectors.

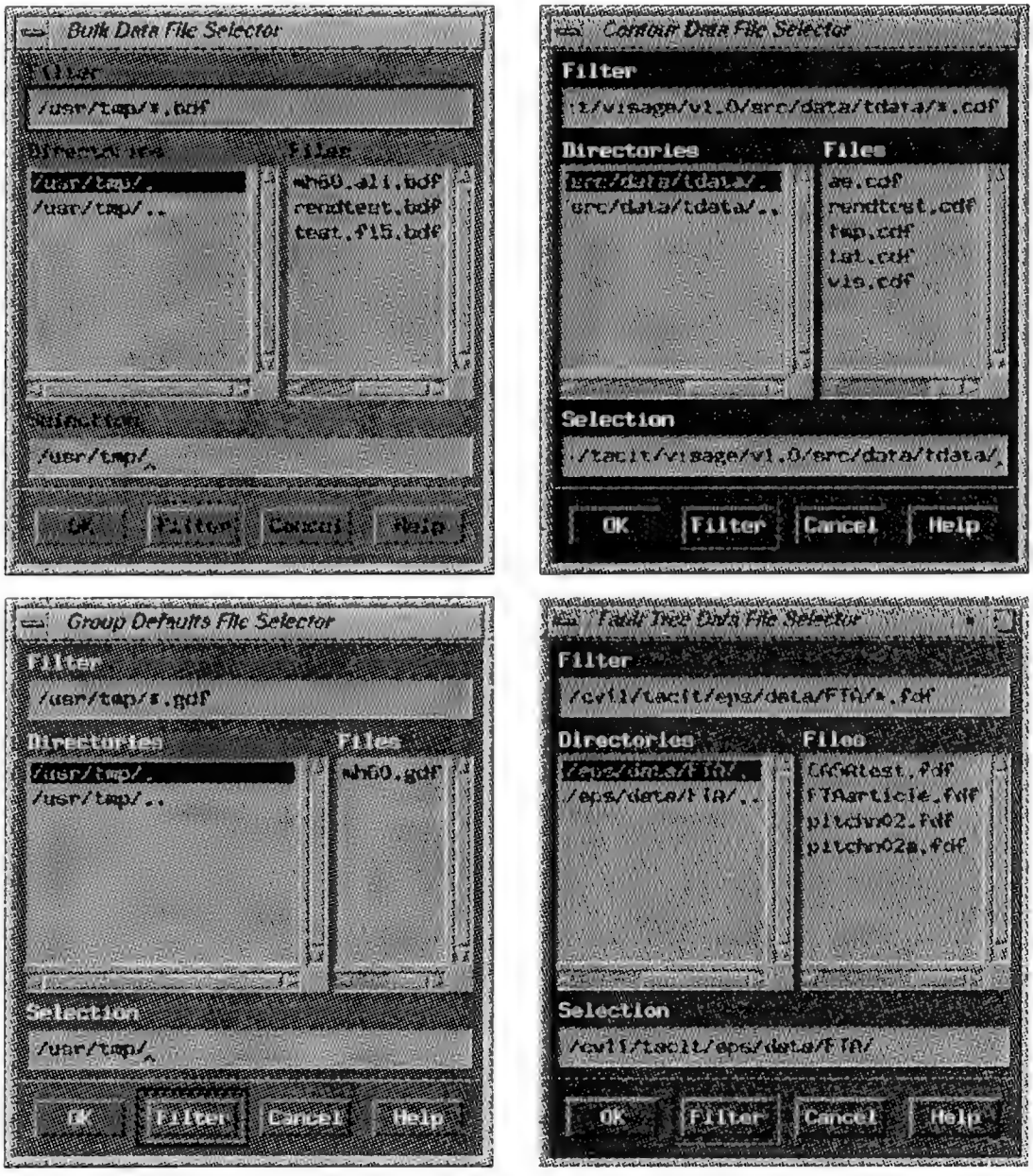
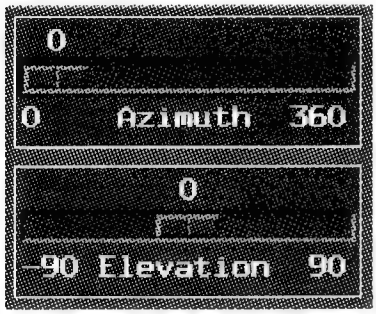
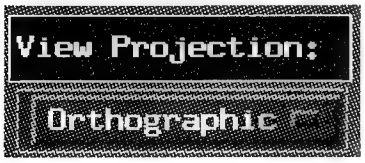


FIGURE 7 File Selectors

2.4 View Controls Reference

The View Controls popup dialog contains menu selections that provide the user with the capability to duplicate any orientation in FASTGEN 4. The initial view orientation is at 0 degrees azimuth, 0 degrees elevation. These selections are described in Table 2.

TABLE 2 View Controls Reference

Item	Effect
Azimuth/Elevation 	Positive azimuthal and elevational rotation in 1 degree increments between 0 and 360 degrees, and-90 and +90 degrees, respectively. The user can directly select an azimuth or elevation by clicking the middle mouse button on the sliding bar path. Clicking on either side of the sliding bar will increment or decrement the value by one. Holding down the left mouse button on top of the sliding bar and dragging will more quickly increment or decrement the value.
Presets (AZ, EL) 	This option menu toggles the *camera view* to preset azimuth and elevation values defined in the *FASTGEN 4.0 User's Manual*. The default is *AZ: 0, EL: 0*.
Orthographic/Perspective 	This option menu toggles the *camera projection* between *perspective* and *orthographic*. The default is *orthographic*.
XYZ Axes 	These toggle the visibility of the coordinate axes in the Display Window off or on. The default for this toggle button is *on*.
ELEMENT Labels 	These toggle the visibility of the ELEMENT labels of the currently visible geometry. The ELEMENT labels appear above the center point of their respective Elements and are shown in dark red. The default for this toggle button is off.

GRID Labels ..... These toggle the visibility of the GRID labels of the currently visible geometry. The GRID labels appear below the center point of their respective Grid points and are shown in dark blue. The default for this toggle button is off.

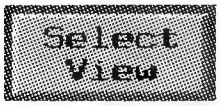
Color Bar ..... Brings up the Color Bar Window, showing the possible color choices for the user in highlighting the various Components and Elements of a target description. On the right side of the Color Bar are the Color Bar index numbers needed by the *Highlight Component(s)* menu selection. These are also the index numbers used in the Contour Data File (see Chapter 3). Underneath the colors of the Color Bar are the user-assigned values which VISAGE reads from the Contour Data File.

Hidden Surfaces ..... This option menu toggles the hidden surfaces algorithms on or off. The Draft option is a quick Z-sort algorithm and will not be a true hidden surface representation when used with complex target descriptions. The Full option provides the best possible hidden surface representation. The default for this option menu is *None*. (NOTE: It is recommended that the user position the target description before turning on the hidden surfaces, to allow the program to save time in redrawing the target description.)

Zoom Out ..... Zoom the current *camera view* out by a factor of 1.5.

Zoom In ..... Zoom the current *camera view* in by a factor of 1.5.

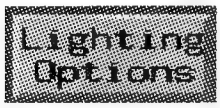
Select View ..... Brings up the Select View popup dialog which allows the user to select a new camera Position, Target, and Viewing Angle. The default Position is *(1.0, 0.0, 0.0)*. The default Target is *(0.0, 0.0, 0.0)*. and the default View Angle is *15.0 degrees*.The view angle determines how much of the scene can be shown. An angle of 90.0 degrees is the maximum. The default value best approximates



what the human eye would perceive. Increasing the view angle effectively moves the camera "back" from the camera target, so that more of the target is visible, similar to the way a zoom lens works. Fish-eye effects are NOT possible.

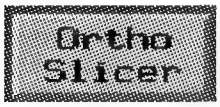
Reset View ..... This selection returns the target description to the initial view (at start-up time). This will not affect any Components or Elements that have been highlighted, or the current visibility of the target description.

Lighting Options ..... Brings up the Rendering Options popup dialog which allows the user to select the lighting and rendering options for the current target description.



Geometry Options ..... Brings up the Geometry Options popup dialog which allows the user to select the color, lighting, transparency, and geometry type (wireframe or solid) options for the current target description.

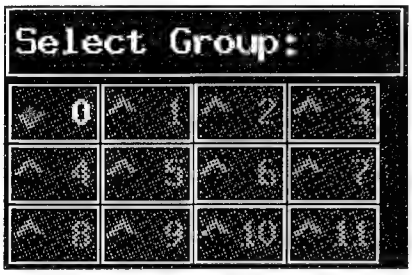
Ortho Slicer ..... Brings up the Orthogonal Slicer popup dialog which allows the user to select from six orthogonal cutting planes to apply to the current target description.



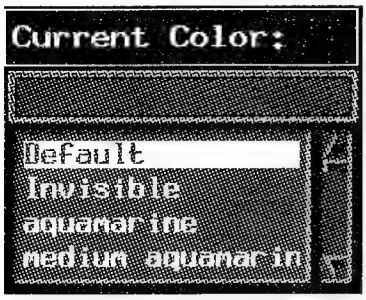
2.5 Geometry Controls Reference

The Geometry Effects Menu contains selections that allow the user to alter the target description's geometric presentation in the Plot Window. These selections are described in Table 3.

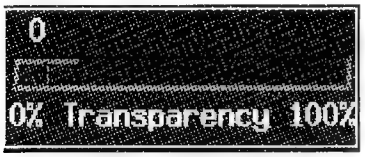
TABLE 3 Geometry Controls Reference

Item	Effect
Set Group Options 	This set of radio buttons allows the user to select which Group's options to display. The *Group(s)* are the Groups in FASTGEN 4. (e.g. aircraft skin, power plant, crew, etc.) This is a dynamic menu in that buttons will only be active for the Group(s) that have been read in by the program from the data file.
Geometry Type 	Change the geometry type of the current Group to a choice of Off, Wireframe, or Solid. The default is that the first Group to be read in will be in *Wireframe* mode. All other Groups will be *Off*.
Group Name 	Displays the currently selected Group's name. This can be edited by the user. All values changed by the user will be retained for the current geometry. The Group Defaults File allows the user to change the default names for the Group(s).
Component Range 	Allows the user to select a Component range in the current Group on which to apply the geometry options. Individual components can be selected by putting the same value in both text fields. The default range is *1 to 999*.

Current Color ..... Displays the currently selected Group's default color. This can be changed by the user by selecting a new color from the scrolled list. All values changed by the user will be retained for the current geometry. The Group Defaults File allows the user to change the default display color of Elements in the particular Groups.

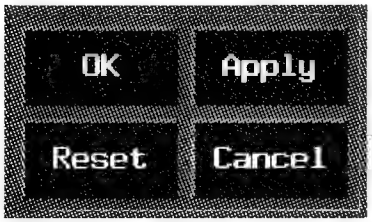


Transparency ..... Allows the user to select the amount of transparency to apply to the current Group/ Component selection. 0% transparency makes the geometry completely opaque. 100% transparency makes the geometry completely transparent. The default is *0%*.



Surface Shading ..... Allows the user to select between one of the following lighting options: None, Flat, Gouraud, and Phong. None indicates that no lighting calculations are to be done on the target description. Flat indicates that lighting should be calculated for each polygon, with no interpolation. Gouraud indicates that lighting should be calculated at each vertex of a polygon and these are interpolated. Phong indicates that lighting should be calculated for a "vertex normal" at each vertex (which is calculated from the surrounding polygon face normals) and these are interpolated. Phong is the best lighting approximation, with Gouraud second and Flat third. The default is *None*.

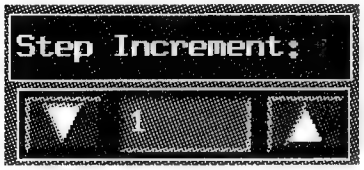
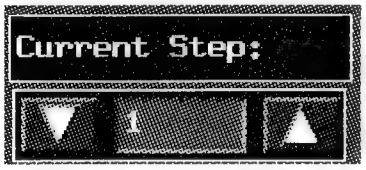
OK/Apply/Reset/Cancel ..... These push-buttons control the use of the Geometry Options. *OK* will apply any changes to the options and dismiss the popup dialog. *Apply* will apply any changes to the options and retain the popup dialog. *Reset* returns the options to their start-up defaults. *Cancel* will reset the options and dismiss the popup dialog.



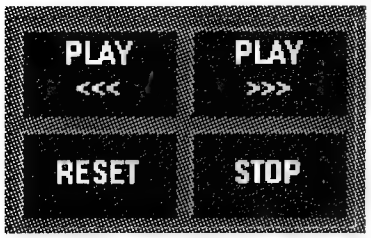
2.6 Animation Controls Reference

The Visual Effects Menu contains selections that allow the user to alter the color schema, visibility, and construction (solid or wireframe) of the target description. These alterations are performed at the Group and Component levels of the target description. These selections are described in Table 4.

TABLE 4 **Animation Controls Reference**

Item	Effect
Step Increment 	Sets the Step Increment for the current Contour Data. This is a range from 1 to 10. Users can decrement or increment the value by clicking on the down or up arrows, respectively. Values can be directly input by typing in the text field (must be terminated by a <return> to be valid).
Current Step 	Sets the Current Step value for the current Contour Data. This is a range from 1 to 500, currently. Users can decrement or increment the value by clicking on the down or up arrows, respectively. Values can be directly input by typing in the text field (must be terminated by a <return> to be valid). Changing this value will result in the data at the new Step being applied to the current target description.
Step Cycle Type 	Determines the cycle of the steps. This can be Linear, Bounce, or Continuous. Linear stepping will stop at the first or last data step, depending on Play direction. Bounce stepping will cycle back and forth between the first and last steps indefinitely. Continuous stepping will loop through the data set, in the Play direction, indefinitely.
Current Value 	This displays the value of the current step, that was assigned in the CDF.

PLAY <<</PLAY >>>/
RESET/STOP..... These push-buttons control the actions of the Animation Controls popup dialog. **PLAY <<<** will start *reverse* play of the data set. **PLAY >>>** will start *forward* play of the data set. **RESET** returns all settings to their original values. **STOP** halts playback of the data set.



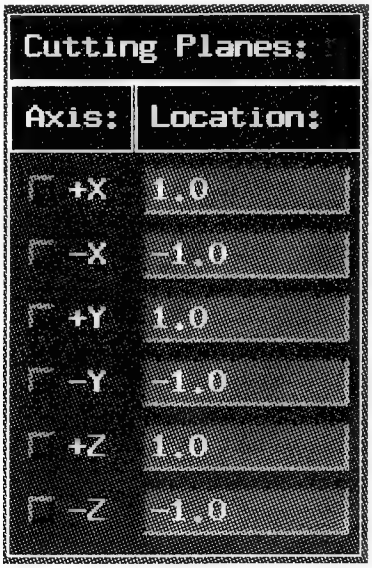
2.7 Orthogonal Slicer Reference

These selections are described in Table 6.

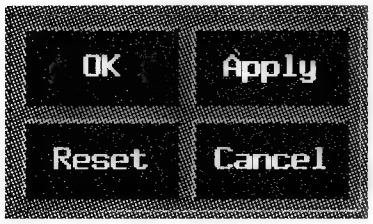
TABLE 5 Orthogonal Slicer Reference

Menu Selection	Effect

Cutting Planes..... These toggle buttons and text fields allow the user to setup 6 orthogonal cutting planes (2 per axis). By activating the toggle buttons, the user can "turn on" a cutting plane. The position along the axis is determined by the associated Location text field. The cutting planes are setup so that geometry on the positive and negative sides of an axis can be "cut away", i.e., the +X plane will "cut away" all geometry on the positive side of its associated Location value and the -X plane will "cut away" all geometry on the negative side of its associated Location value. The same holds for the +Y, -Y, +Z, and -Z axes. To "turn off" an existing cutting plane, simply disable its toggle button. The default is that all cutting planes are disabled.



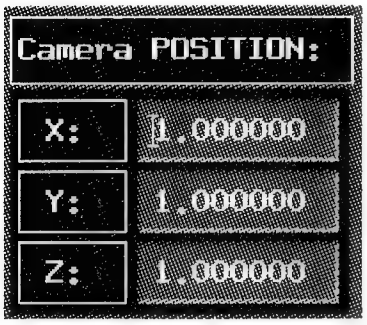
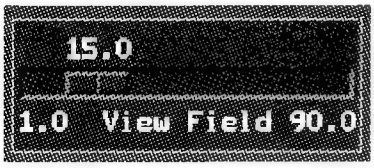
OK/Apply/Reset/Cancel..... These push-buttons control the use of the Orthogonal Slicer. *OK* will apply any changes to the options and dismiss the popup dialog. *Apply* will apply any changes to the options and retain the popup dialog. *Reset* returns the options to their start-up defaults. *Cancel* will reset the options and dismiss the popup dialog.



2.8 Select View Reference

These selections are described in Table 7.

TABLE 6 Select View Reference

Menu Selection	Effect
Camera POSITION 	These text fields allow the user to setup the camera position by specifying the XYZ coordinates. The default coordinates are *(1.0, 1.0, 1.0)*.
Camera TARGET 	These text fields allow the user to setup the camera target by specifying the XYZ coordinates. The default coordinates are *(0.0, 0.0, 0.0)*.
Camera POSITION 	These text fields allow the user to setup the camera view field by specifying the angle. The default angle is *15.0*.
OK/Apply/Reset/Cancel 	These push-buttons control the use of the Select View dialog. *OK* will apply any changes to the options and dismiss the popup dialog. *Apply* will apply any changes to the options and retain the popup dialog. *Reset* returns the options to their start-up defaults. *Cancel* will reset the options and dismiss the popup dialog.

2.9 Basic Editor Reference

The Basic Editor is composed of two popup dialogs:

Data Card Editor

Data Viewer

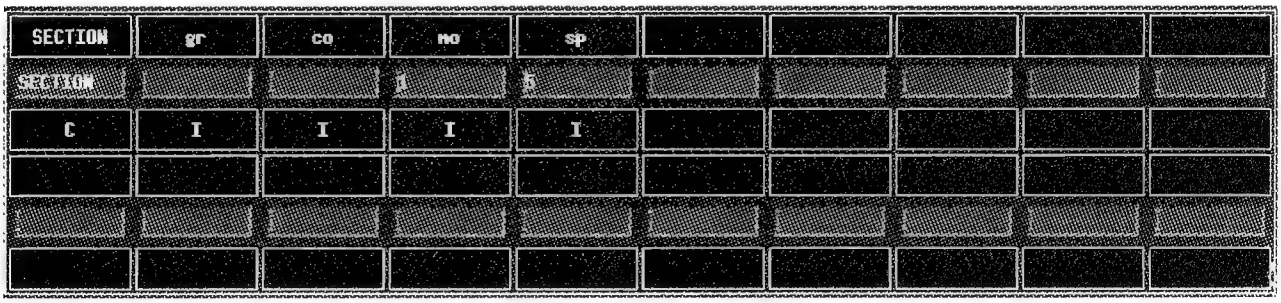
The Data Card Editor (DCE) is a template mechanism for creating the various card types found in the BDF. In future versions, this will be expanded to include the CDF, GDF, and Case Control deck. The DCE is setup to resemble the card description pages of the FASTGEN 4 User's Manual. The user is expected to be familiar with the terminology used by that manual.

The Data Viewer (DV) allows the user to view and/or edit a single component at a time, in conjunction with the use of the DCE. The DCE can be used to create new cards in the current component or the user may create new entries, or edit or copy existing ones in the DV.

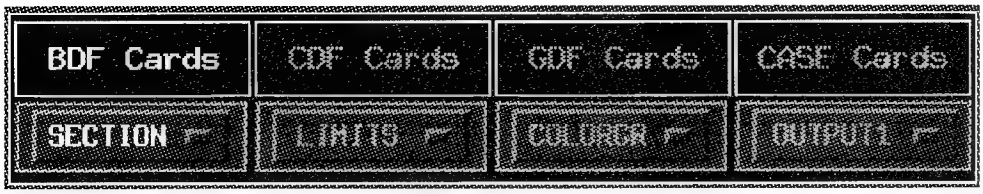
The selections for the DCE are described in Table 8 and the selections for the DV are described in Table 9.

TABLE 7 **Data Card Editor Reference**

Menu Selection	Effect



Card Template ..... These text fields allow the user to create any card found in the BDF. The *BDF Cards* pulldown option menu selection allows the user to select which card they would like to edit. The <Tab> and <Shift><Tab> keys allow the user to move forward and back, respectively, in the active text fields and any changes are automatically updated to the DV.



BDF Cards Menu..... This pulldown menu allows the user to load any card found in the BDF into the *Card Template*. The CDF, GDF, and Case Cards are for future versions.

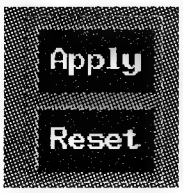
Editing.....

These radio buttons allow the user to determine which part of the target description, if it exists, should be loaded into the DV. These radio buttons work in conjunction with the *Group/Component* text fields. *New* will create a new component. *Existing* will load an existing component in the DV. *Header* will load the header portion, if any, in the DV. If the user selects *New* and specifies an existing component in the *Group/Component* text fields, then the *Existing* button will be selected and the component loaded into the DV. If the user selects existing and specifies a non-existent component, then the *New* button will be selected and the DV will be cleared.

Group/Component Selection.....

These text fields allow the user to determine which component of the target description, if it exists, should be loaded into the DV. These text fields work in conjunction with the Editing radio buttons. To specify the *Group* a user only has to type in text field. However, to specify the *Component* the user must type in <Enter> (or <Return>) after the typing in the number to activate the DV.

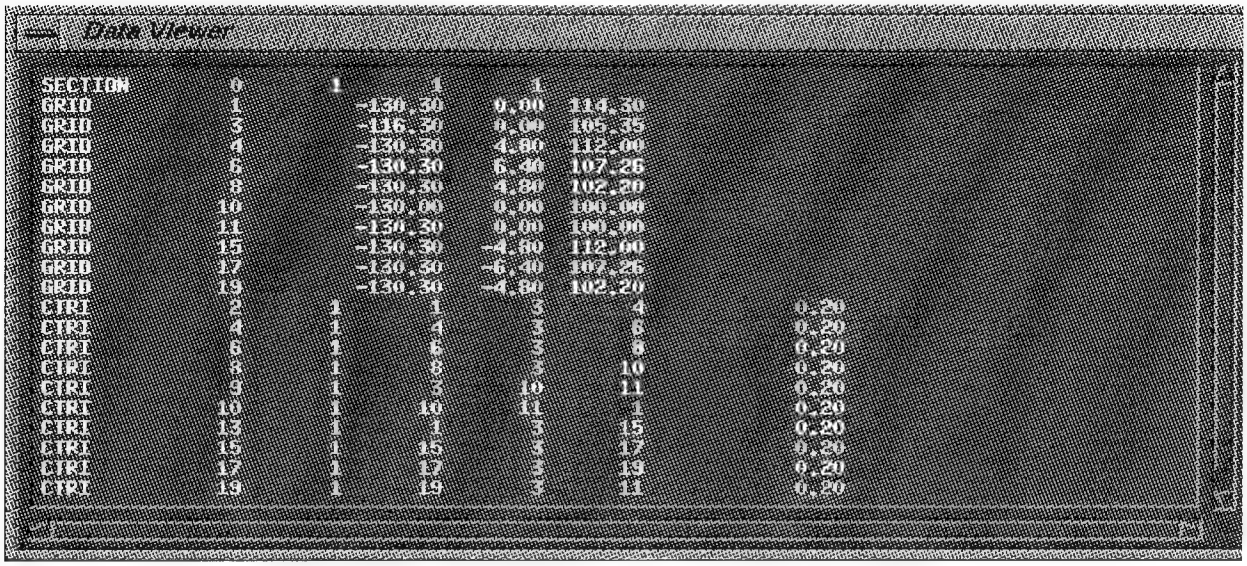
Apply/Reset.....



These push buttons allow the user to *Apply* any changes to the DV or to *Reset* the current card selection to its default values.

TABLE 8 Data Viewer Reference

Menu Selection	Effect



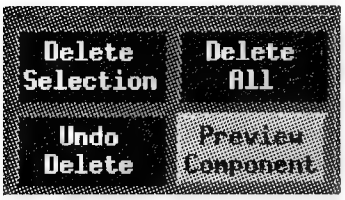
Data View Area..... This text area contains the current component being edited. The user may edit cards directly or use the DCE to create new ones. The mouse functions for the Data View Area are as follows:

Left: hold and drag to select text
single-click to select cursor position
double-click to select a word
triple-click to select a line
quadruple-click to select all

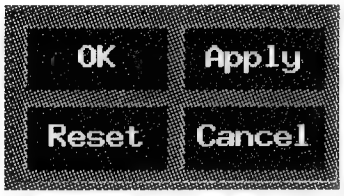
Middle: single-click to copy selected text
hold down over selected text and drag to move text

Right: single-click to paste last selected X-window text

Delete Selection/
Delete All/
Undo Delete/
Preview Component..... These push buttons allow the user to have an Undo capability and to preview the current data. By using the *Delete Selection* and *Delete All* buttons instead of the <Backspace> button, the user has a last-action undo capability. The *Preview Component* button allows the user to see what changes will look like without actually changing the current data. All previewed geometry will be shown with black faces and red edges to distinguish it from the original data.



OK/Apply/Reset/Cancel..... These push-buttons control the use of the Data Card Editor and Data Viewer dialogs. *OK* will apply any changes to the target description and dismiss the popup dialogs. *Apply* will apply any changes to the target description and retain the popup dialogs. *Reset* returns the dialogs to their start-up defaults. *Cancel* will reset and dismiss the popup dialogs.

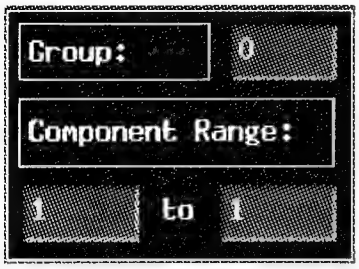
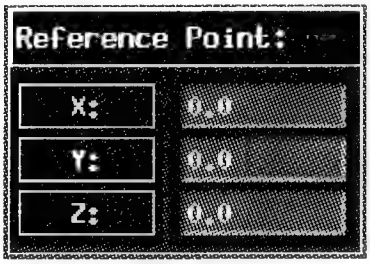
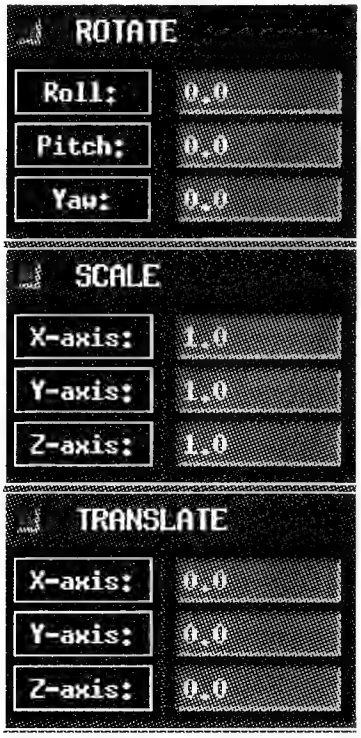


2.10 Advanced Editor Reference

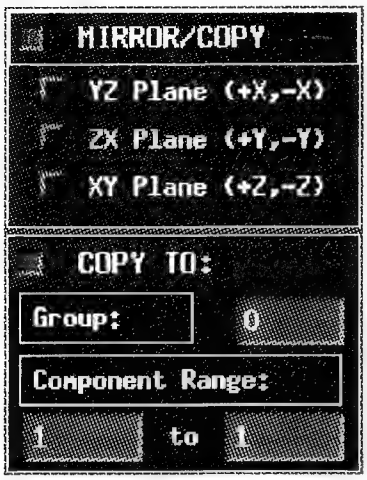
The Advanced Editor (AE) allows the user to apply transformations to single or multiple Components. It also allows the user to copy single or multiple Components, either in the same Group (space permitting) or to another Group. (NOTE: By default, Scaling is limited to equal values for each axes. The user must select independent axis scaling by selecting Independent Scaling under the Options menu on the menu bar.)

The selections for the AE are described in Table 10.

TABLE 9 Advanced Editor Reference

Menu Selection	Effect
Group/Component Range 	These text fields allow the user to specify the Group and range of Components to which to apply the transformations.
Reference Point 	These text fields allow the user to specify a reference point for use in calculating the Rotate and Scale transformations. This point does NOT apply to any other transformations.
ROTATE/SCALE/ TRANSLATE 	These toggle buttons and text fields allow the user to specify the rotation, scaling, and translation transformations to apply to the specified component(s). The user can select any combination of the three, but the transformations will always be applied in the following order: Rotate, Scale, Translate.

MIRROR/COPY & COPY TO

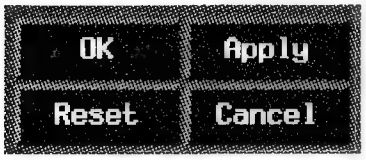


These toggle buttons and text fields allow the user to specify how to mirror the specified component(s) and copy to the specified Group and Component(s).A mirror operation requires a Copy operation, so the two functions are linked. The user can also specify a Copy operation without doing a Mirror operation.

Delete Component & Preview Component

These push buttons allow the user to Delete a component and to Preview any transformations. The Delete button will delete the component(s) specified in the Group and Component Range text fields. The Preview button allows the user to check if the transformations are being applied as expected.

OK/Apply/Reset/Cancel



These push-buttons control the use of the Advanced Editor dialog. *OK* will apply any changes to the target description and dismiss the popup dialog. *Apply* will apply any changes to the target description and retain the popup dialog. *Reset* returns the dialog to its start-up defaults. *Cancel* will reset and dismiss the popup dialog.

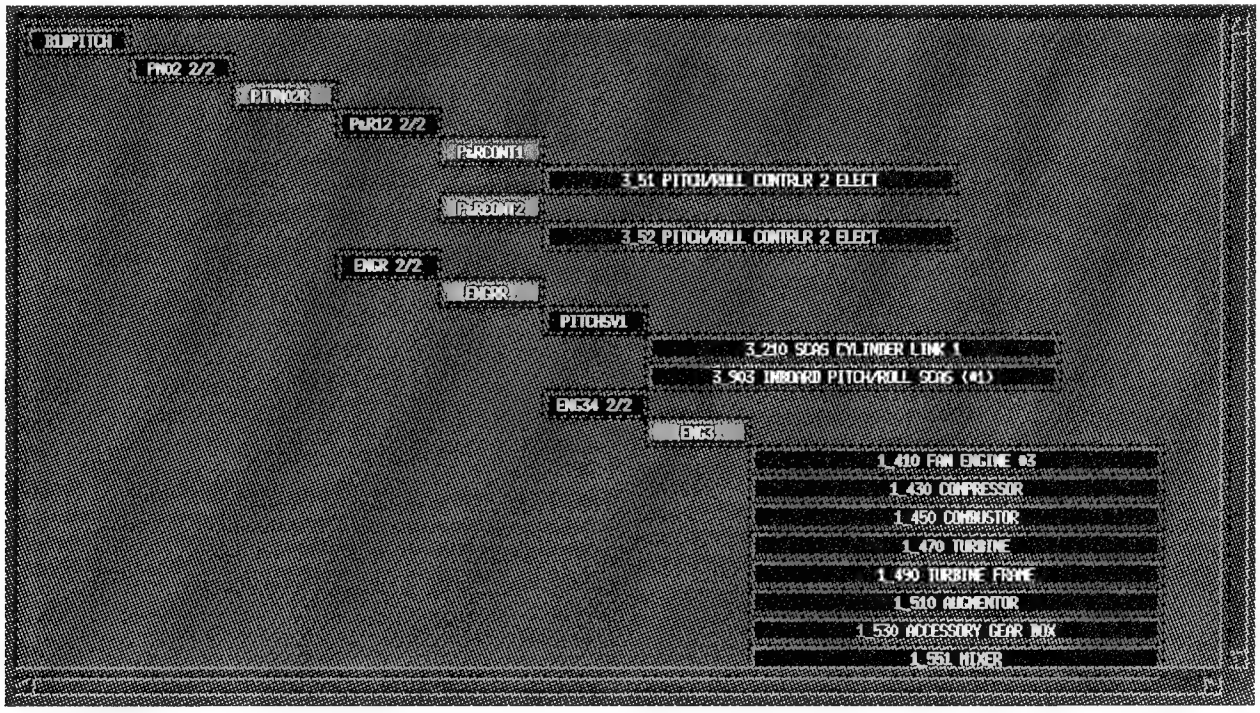
2.11 Fault Tree Data Viewer Reference

The Fault Tree Data Viewer allows the user to view fault trees produced by COVART 4 in an outline format. This format is used to simplify the display of the fault tree. The Fault Tree Data Viewer also links the fault tree to the target description by the components defined in the Fault Tree Data File (FDF). See Ch. 3.5 for a more detailed description of the FDF.

The usage of the Fault Tree Data Viewer is described in Table 10.

TABLE 10 **Fault Tree Data Viewer Reference**

Menu Selection	Effect



Fault Tree Display ..... This scrolled window holds the push-buttons that link the fault tree to the FASTGEN target description. The user can highlight the associated target description components by clicking on the push-buttons.

Reset Display ..... Resets the Fault Tree Display to the start-up condition of no fault tree components highlighted in the target description.

2.12 Mouse Functions

Certain areas of the VISAGE GUI have functions tied to the mouse selection buttons. A standard 3-button mouse is assumed.

The Display Window area of the GUI has 3 functions tied to the mouse buttons. These 3 functions are:

1] Select Geometry

2] Select Camera Target

3] Quick Zoom

2.12.1 Select Geometry

Select Geometry allows the user to select any visible geometry element in the Display Window and its Element ID information will be displayed in the VISAGE information area. The target description must be displayed in Draft or Full hidden surfaces to enable this action. If not, an error message will be displayed in the information area. This function is tied to the *left mouse button*.

2.12.2 Select Camera Target

Select Camera Target allows the user to designate a new camera target. By selecting a point in the Display Window, VISAGE will effectively turn that point into the new camera target, which will become the approximate center of the new view. This function is tied to the *middle mouse button*.

2.12.3 Quick Zoom

Quick Zoom allows the user to "zoom" the camera view into or out of an area of the currently displayed geometry. By selecting a point in the Display Window, VISAGE will use that point as the new camera target and zoom in by a factor of 2. This function is tied to the *right mouse button*.

To zoom in, the user can click once with the right mouse button at the desired position.

To zoom out, the user must hold down the <shift> button and click once with the right mouse button at the desired position.

Note: The **Zoom Out** and **Zoom In** buttons should be used for fine adjustments of the camera view.

3.0 Data File Formats

3.1 Introduction

VISAGE expects data files of four possible formats: the FASTGEN 4 bulk data deck format (BDF), the VISAGE Contour Data File (CDF), the VISAGE Group Defaults File (GDF), and the VISAGE Fault Tree Data File (FDF).

3.2 VISAGE Data Deck

VISAGE in its current implementation supports the following FASTGEN 4 bulk data deck cards:

SECTION	**CCONE1**	**CCONE2**
GRID	**CHEX1**	**CHEX2**
ENDDATA	**CLINE**	**CQUAD**
$COMMENT	**CSPHERE**	**CTRI**

The following data deck cards are not found in the FASTGEN 4 bulk data deck. They are extensions for endgame analysis support:

POINT	**UNITGBL**	**UNITLCL**
CORDAE		

Also, files using the following NASTRAN bulk data deck cards can be viewed:

GRID	**CTRIA1**	**CTRIA2**
CTRIA4	**CQUAD1**	**CQUAD2**
CQUAD4	**CHEXA1**	**CHEXA2**

with the addition of SECTION and ENDDATA cards at the beginning and end of the file, respectively.

Please refer to the ***FASTGEN 4.0 Users' Manual*** for the FASTGEN 4 - NASTRAN cross reference, and for more information on these cards. Currently, all unsupported cards are treated as non-valid and are ignored. Likewise, an incorrectly spelled valid "card" is also ignored.

Appendix A shows an example BDF.

The following is a breakdown of the valid Groups:

Group	Description
0-9	Target bulk data
10	Attacker bulk data (The missile and fuze cone need to be in separate components)
11	Fragments, although POINT "cards" may appear in any Group, this is reserved for point data only. Each Time Step should be a separate component.

Figure 8 illustrates the order dependencies of the VISAGE bulk data deck.

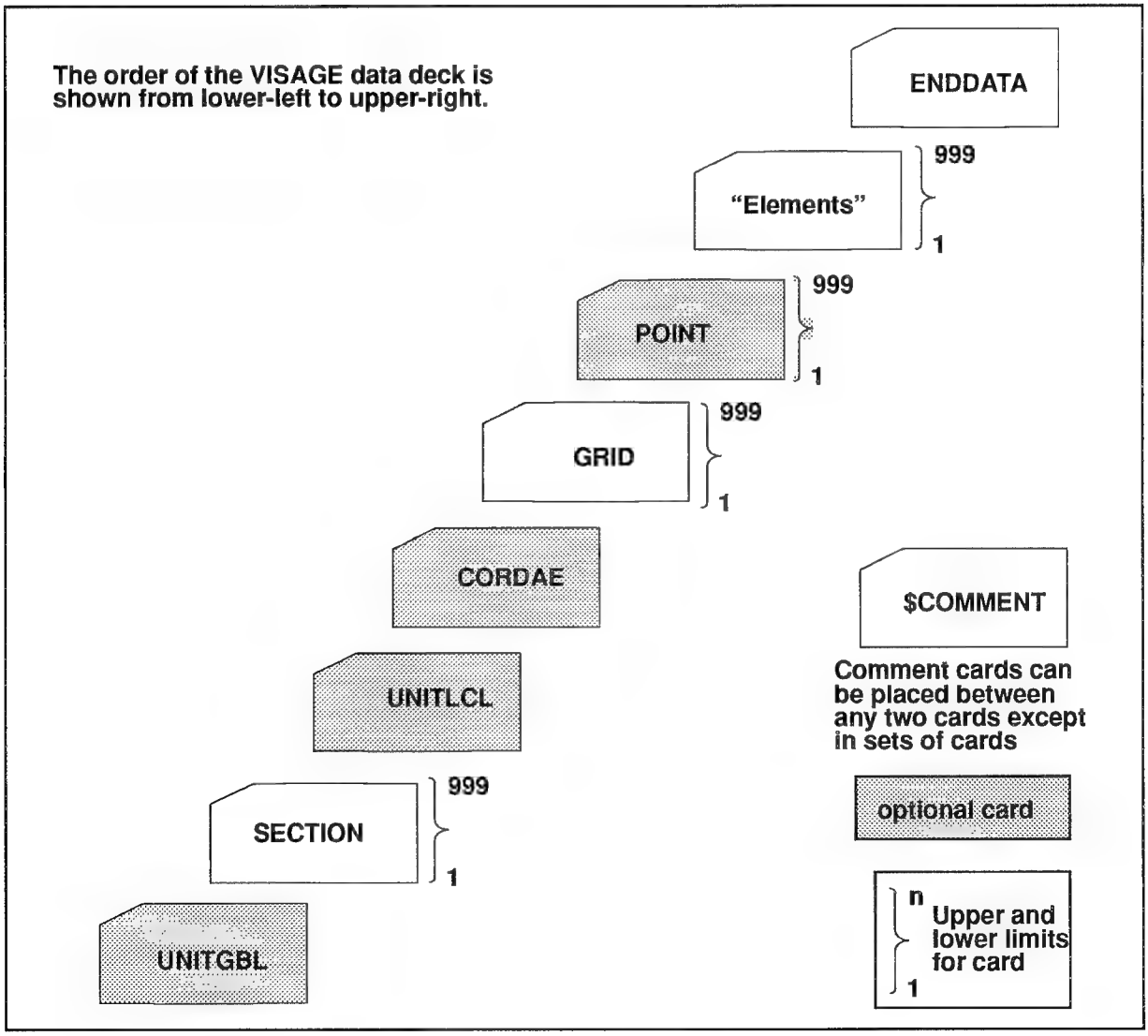


FIGURE 8 VISAGE Data Deck Order

Bulk Data Deck Card: CORDAE

Description:

Allows the user to translate Component(s) and to apply yaw, pitch, and roll rotations to the Component(s). This "card" is used in conjunction with one or two POINT cards, which must follow immediately after the CORDAE "card". These POINT cards are part of the 10,000 POINT per SECTION limit. This "card" is optional.

Format, Example, and Data Type:

Field 1	Field 2	Field 3	Field 4	Field 5	Field 6	Field 7	Field 8	Field 9	Field 10
CORDAE	cid	rpid	tpid	yaw	pitch	roll			
CORDAE	1	0	777	0.0	90.0	0.0			
string	integer	integer	integer	real	real	real			

Field	Contents
cid	Component ID number (integer > 0).
rpid	Reference POINT ID (integer >=0). This references a POINT "card" that provides the reference point on the Component to use for any translations or rotations.
tpid	Translation POINT ID (integer > 0). This references a POINT "card" that provides the point to translate the Component to. If RPID = 0, then the x,y,z values of the TPID POINT "card" are used as translational displacement distances (Δx, Δy, Δz).
yaw	Value for rotation about a Component's local Z-axis (real).
pitch	Value for rotation about a Component's local Y-axis (real).
roll	Value for rotation about a Component's local X-axis (real).
Remarks:	The CORDAE "card" expects at least one POINT "card", which must immediately follow the CORDAE "card", if not VISAGE will ignore the "card", print a warning, and proceed.
	All rotations and/or translations are based on the "right-hand rule".

Bulk Data Deck Card: POINT

Description:

User defines an x,y,x location in space representing a point. The VIS field is used to distinguish whether a POINT "card" is to visible or not. This is important when using the POINT "card" in conjunction with the CORDAE "card". This "card" is optional.

Format, Example, and Data Type:

Field 1	Field 2	Field 3	Field 4	Field 5	Field 6	Field 7	Field 8	Field 9	Field 10
POINT	pid	vis	x	y	z				
POINT	15	0	29.82	236.32	138.55				
string	integer	integer	real	real	real				

Field	Contents
pid	POINT ID number (integer > 0). 1 to 1000 reserved for POINT cards used with VIEW cards > 1000 reserved for all other POINT cards
vis	POINT Visibility flag (0 <= integer <= 5). 0 = Visible Reference Point 1 = Invisible Reference Point 2 = Visible Non-Hit 3 = Invisible Non-Hit 4 = Visible Hit 5 = Invisible Hit
x	Point position on X axis (real).
y	Point position on Y axis (real).
z	Point position on Z axis (real).

Remarks: The POINT "card" is for introducing point data and is not part of the FASTGEN 4 bulk data deck. POINTs are treated as any other Elements (e.g. CQUAD, CTRI, etc.). POINT cards used with VIEW cards must have a unique PID number between 1 and 1000. NOTE: Each SECTION can have no more than 10,000 POINT cards.

Bulk Data Deck Card: UNITGBL

Description:

Global UNITs "card". The user provides a divisor to be used in scaling all GRID and POINT "card" data in the "deck". This "card" must be before the first SECTION "card". This "card" is optional and VISAGE will default to 1.0 in its absence.

Format, Example, and Data Type:

Field 1	Field 2	Field 3	Field 4	Field 5	Field 6	Field 7	Field 8	Field 9	Field 10
UNITGBL	uval								
UNITGBL	12.0								
string	real								

Field	Contents
uval	Value to be used as divisor for data (real).

Remarks: The UNITGBL "card" must appear before the first SECTION card in a file. VISAGE will ignore it if it appears elsewhere in a file and use the default value.

UNITLCL cards will override the UNITGBL value (either the user-specified value or the default value). If the UNITLCL "card" is specified, then that Component will use the UNITLCL divisor and all succeeding Components will use the UNITGBL divisor.

Bulk Data Deck Card: UNITLCL

Description:

Local UNITs "card". User provides a divisor to be used in scaling all GRID and POINT "card" data in the current SECTION. This "card" is optional.

Format, Example, and Data Type:

Field 1	Field 2	Field 3	Field 4	Field 5	Field 6	Field 7	Field 8	Field 9	Field 10
UNITLCL	uval								
UNITLCL	12.0								
string	real								

Field	Contents
uval	Value to be used as divisor for data (real).

Remarks: UNITLCL cards will override the UNITGBL value (either the user-specified value or the default value). If the UNITLCL "card" is specified, then that Component will use the UNITLCL divisor and all succeeding Components will use the UNITGBL divisor.

3.3 Contour Data File (CDF)

The CDF is an optional file that allows the user to redefine the color schema of individual Elements of the target description drawn from the bulk data deck. It also allows the user to assign values to the chosen colors, which will be displayed on the Color Bar potion of the VISAGE window. Any colors not used in the CDF will remain without values. Appendix B shows an example CDF.

VISAGE will override any mistakes in the type (i.e. real or integer) of the entries, but it cannot correct any subsequent errors resulting from these mistakes, so if an incorrect color is displayed or the color's assigned value is incorrect, check the CDF for entry value typecast errors.

(NOTE: All Contour Data File filenames should end with an extension of ".cdf")

Figure 9 illustrates the order dependencies of the VISAGE Contour Data File.

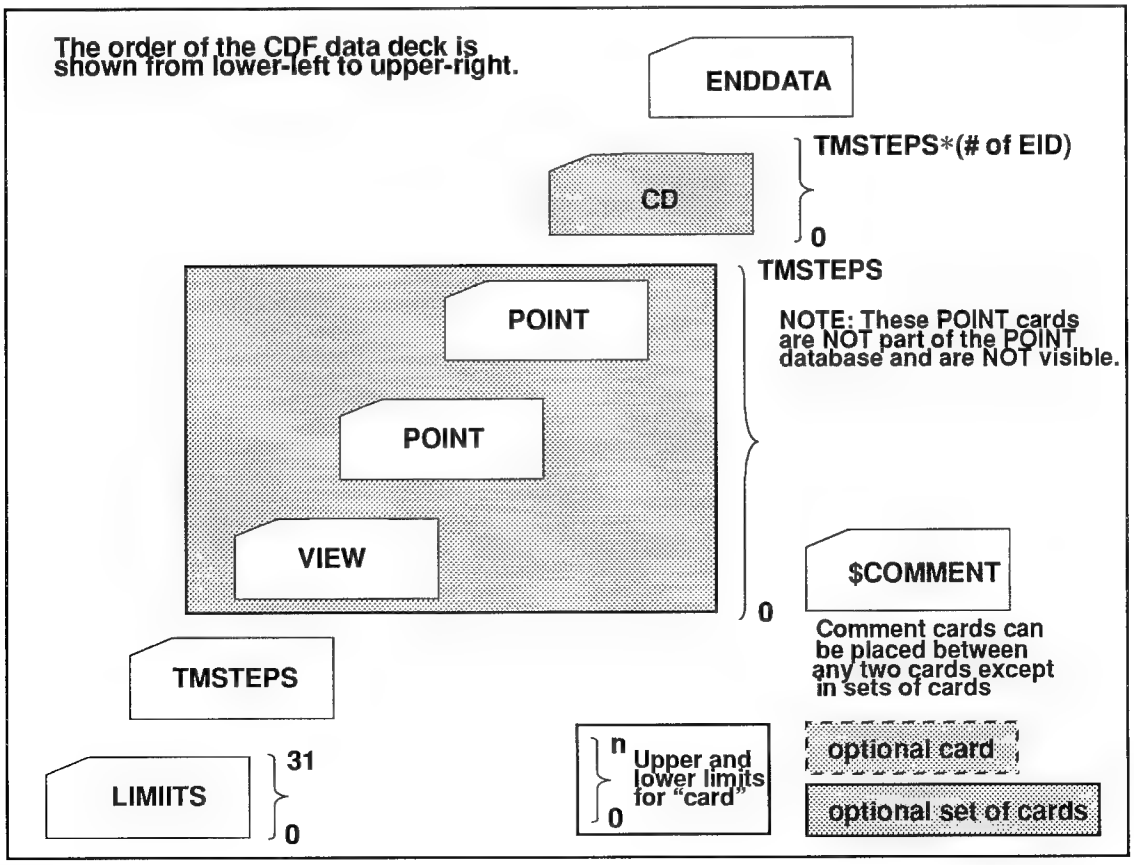


FIGURE 9 CDF Data Deck Order

Contour Data File Card: CD

Description:

Contour data. This "card" sets the color for the specified Element or Component, at the specified time step. The number of CD cards is limited by the number of Elements in the target description(s). CD cards can be used in conjunction with VIEW cards, or by themselves. If VIEW cards are used, the TIME value for the VIEW cards must correspond to those used in the CD cards.

Format, Example, and Data Type:

Field 1	Field 2	Field 3	Field 4	Field 5	Field 6	Field 7	Field 8	Field 9	Field 10
CD	gr	co	eid	time	mag				
CD	0	1	1	1.00	0.24				
string	integer	integer	integer	real	real				

Field	Contents
gr	Group ID number (0 <= integer <= 11).
co	Component ID number (integer > 0).
eid	Element ID number (integer < 0 (Component) or integer > 0 (Element)).
time	Time step value (real).
mag	Magnitude value (real).

Remarks: By specifying a NEGATIVE value for the Element ID number, VISAGE will set the color of the entire Component, rather than just a specific Element.

All Elements with time steps must be defined for all time steps, otherwise the non-defined Elements will default to the last color specified.

Contour Data File Card: LIMITS

Description:

User defines the limits of magnitude for individual color bar values. (Maximum of 32 LIMITS cards.)

Format, Example, and Data Type:

Field 1	Field 2	Field 3	Field 4	Field 5	Field 6	Field 7	Field 8	Field 9	Field 10
LIMITS	cbid	max	min						
LIMITS	1	1.0	0.0						
string	integer	real	real						

Field	Contents
cbid	Color bar ID number (0 <= integer <= 31).
max	Maximum magnitude value (real).
min	Minimum magnitude value (real).

Remarks: The following is a list of the approximate color ranges and the corresponding CBIDS:

Color Range	CBIDs
Red (5 shades)	0 - 4
Orange (5 shades)	5 - 9
Yellow (5 shades)	10 - 14
Green (5 shades)	15 - 19
Blue (5 shades)	20 - 24
Purple (5 shades)	25 - 29
Invisible	30
Default Group Color	31

Contour Data File Card: TMSTEPS

Description:

User provides the number of time steps in the CDF and sets the color bar title. This "card" must be used if CD or VIEW cards are used.

Format, Example, and Data Type:

Field 1	Field 2	Field 3	Field 4	Field 5	Field 6	Field 7	Field 8	Field 9	Field 10
TMSTEPS	num	text							
TMSTEPS	2	color	bar title						
string	integer	16-char	string						

Field	Contents
num	Number of time steps in CDF (1 <= integer <= 500).
text	Title to be used for the color bar window (maximum 16 character string).

Contour Data File Card: VIEW

Description:

User can provide camera-to-target viewpoint for each time step. By specifying a camera position, a camera target, the time, and the view angle, the user can have progressive viewpoint changes. This "card" must be followed by two POINT cards, which will designate the camera position (CPPID) and the camera target (CTPID). VIEW cards can be used in conjunction with CD cards, or by themselves. If CD cards are used, the TIME value for the VIEW cards must correspond to those used in the CD cards.

Format, Example, and Data Type:

Field 1	Field 2	Field 3	Field 4	Field 5	Field 6	Field 7	Field 8	Field 9	Field 10
VIEW		cppid	ctpid	time	vangle				
VIEW		444	555	1.00	15.0				
string		integer	integer	real	real				

Field	Contents
cppid	Camera Position POINT ID (1 <= integer <= 1000).
ctpid	Camera Target POINT ID (1 <= integer <= 1000).
time	Time step value (real).
vangle	View Angle value (real > 0.0).

Remarks: The POINTs used with this "card" are not visible and will not be part of the POINT database. Also, the two POINT cards **must** have unique POINT ID numbers between 1 and 1000.

3.4 Group Defaults File (GDF)

The GDF is an optional file that allows the user to redefine the name and the default colors of the Groups of a target description drawn from the bulk data deck. The *COLORGR* and *GRPNAME* cards allow the user to change the default Group names and default display colors of a target description. This allows for multiple target descriptions to be displayed from the same data set. Appendix C shows an example GDF.

The following is a breakdown of the default colors of the Groups:

Group	Default Color	Default Name
0	Grey	Skin
1	Salmon	Power Plant
2	Gold	Crew
3	Yellow Green	Controls
4	Cornflower	Fuel System
5	Yellow	Ammunition
6	Sky Blue	Armaments
7	Maize	Structure
8	Olive Green	Electrical
9	Lavender	Miscellaneous
10	Cadet Blue	Missile

(NOTE: All Group Defaults File filenames should end with an extension of ".gdf")

Figure 10 illustrates the order dependencies of the VISAGE Group Defaults File.

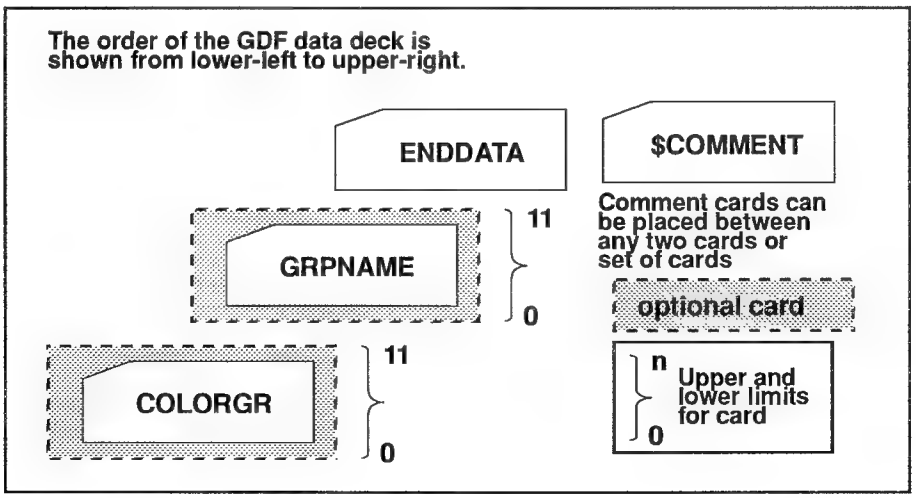


FIGURE 10 GDF Data Deck Order

Group Defaults File Card: COLORGR

Description:

User can define the default display color of a Group.

Format, Example, and Data Type:

Field 1	Field 2	Field 3	Field 4	Field 5	Field 6	Field 7	Field 8	Field 9	Field 10
COLORGR	gr	coid							
COLORGR	0	55							
string	integer	integer							

Field	Contents
gr	Group ID number (0 <= integer <= 11).
coid	Color ID number (0 <= integer <= 63).

Remarks: On the following page is a list of the available colors and the corresponding COIDS:

Color Name	COID
aquamarine	0
med. aquamarine	1
black	2
blue	3
cadet blue	4
cornflower blue	5
dark slate blue	6
light blue	7
light steel blue	8
med. blue	9
med. slate blue	10
midnight blue	11
navy blue	12
sky blue	13
slate blue	14
steel blue	15
coral	16
cyan	17
firebrick	18
gold	19
goldenrod	20

Color Name	COID
dark goldenrod	21
green	22
dark green	23
dark olive green	24
forest green	25
lime green	26
med. sea green	27
med. spring green	28
olive drab	29
pale green	30
sea green	31
spring green	32
yellow green	33
dark slate grey	34
dim grey	35
light grey	36
dark khaki	37
magenta	38
maroon	39
orange	40
orchid	41

Color Name	COID
dark orchid	42
med. orchid	43
pink	44
plum	45
red	46
indian red	47
pale violet red	48
orange red	49
violet red	50
salmon	51
sienna	52
tan	53
thistle	54
turquoise	55
dark turquoise	56
med. turquoise	57
violet	58
blue violet	59
wheat	60
white	61
yellow	62
green yellow	63

Group Defaults File Card: GRPNAME

Description:

User can define Group "name" to be displayed on the Visual Effects menu target description type buttons (wireframe/solid).

Format, Example, and Data Type:

Field 1	Field 2	Field 3	Field 4	Field 5	Field 6	Field 7	Field 8	Field 9	Field 10
GRPNAME	gr	text							
GRPNAME	0	group	name						
string	integer	16-char	string						

Field	Contents
gr	Group ID number (0 <= integer <= 11).
text	Text to be substituted for the default Group name (maximum 16-character string).

3.5 Fault Tree Data File (FDF)

The FDF is an optional file that allows the user to display fault tree data. This fault tree data may have come from the COVART 4 program or the user may have created his own tree data file.

A fault tree is a graphical representation of a Boolean equation of a failure event. A fault tree consists of elements arranged in serial and/ or parallel fashion. Respectively, the serial and parallel arrangements are equivalent to the Boolean OR and AND operations. In a simple serial arrangement, at least one element must fail for the entire construct to fail. In a simple parallel arrangement, one element in each path must fail for the entire construct to fail. A hybrid arrangement, containing both serial and parallel constructs, will fail according to the combined effect of all the constructs.

The usual graphical representation of a fault tree is a block diagram. An advantage of this is that the relationships between elements are readily seen. A disadvantage is that, for large fault trees, the representation may not be completely displayed on a computer screen and may take many pages to print out. Therefore, it was necessary to determine a display format capable of being represented by combinations of low-level Motif widgets while still adequately conveying the tree-like nature of the data.

To simply fulfill these requirements, an outline display format was selected. The outline format is preferable for several reasons. First, it is easily created using only push-button widgets, since no connecting lines are required to delineate the tree structure. Second, the push-button callback functions enable the highlighting of the associated fault tree components in the corresponding target description. Third, it is easily input using the strictly ordered format. Fourth, the tree can be easily rendered, as each card is read in, without having to recompute the entry arrangements. Fifth, the outline format allows for hardcopy output of the tree data in a form that is almost as intuitive as the tree format itself and is easier to produce than tree diagrams.

The data format is strictly ordered to simplify the input subroutines. Basically, the format consists of several conceptual levels; the Set, Function, System, Subsystem, and Component Levels. Each of these levels is designated by formatted data entries referred to as cards. These levels represent the component parts of the tree; the various sub-trees and the leaf (end) nodes.

An input file is produced by traversing a pre-defined fault tree in a depth-first manner and each entry is designated by the appropriate

card, based on the entry's position in the tree. In the outline format, each row may contain only one entry. Entries are designated by the FTSET, FTFUNC, FTSYS, FTSUB, FTCOMP, and $CNAME cards, which are described on the following pages. Appendix D shows an example FDF.

VISAGE will override any mistakes in the type (i.e. real or integer) of the entries, but it cannot correct any subsequent errors resulting from these mistakes, so if an incorrect value or text description is seen, check the FDF for entry value typecast errors.

(NOTE: All Fault Tree Data File filenames should end with an extension of ".fdf")

Figure 11 illustrates the order dependencies of the VISAGE Fault Tree Data File.

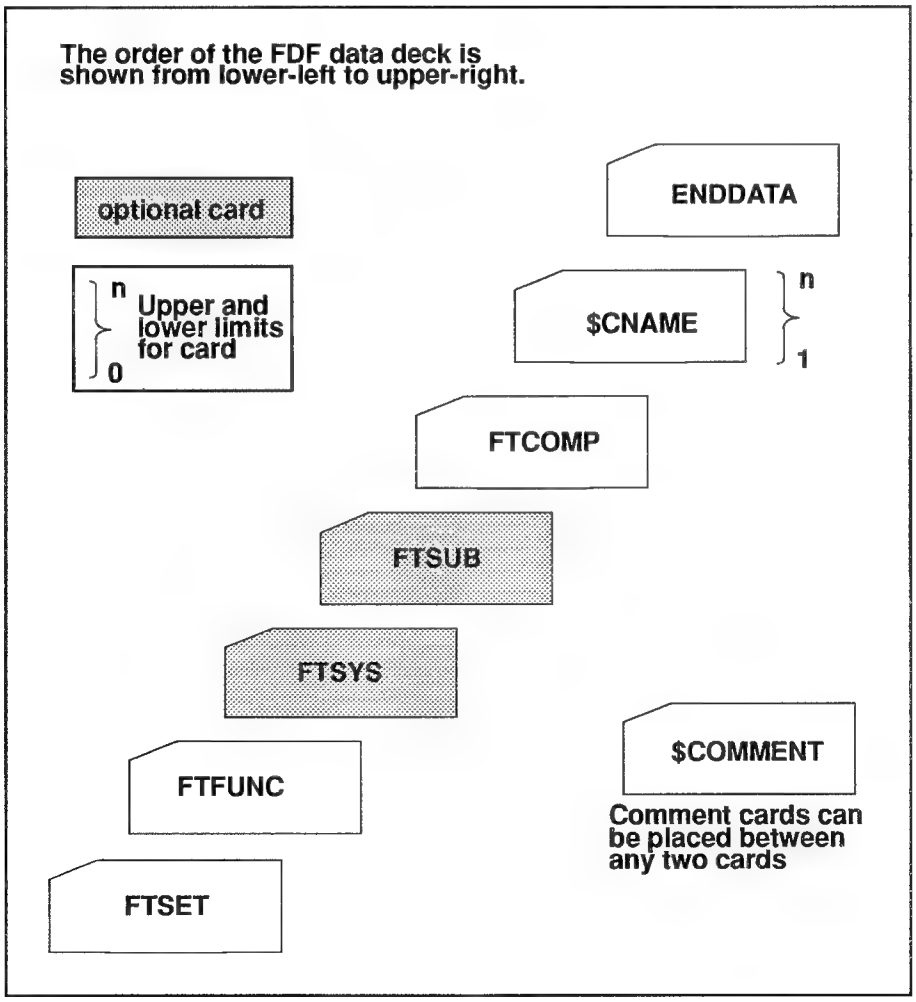


FIGURE 11 FDF Data Deck Order

Fault Tree Data File Card: FTSET

Description:

The FTSET card allows the user to specify a name for the set and how many functions it contains. Each set represents the top level of a particular tree. Generally, there will only be one set (or tree) per input file, but there is no limitation on the number of sets.

Format, Example, and Data Type:

Field 1	Field 2	Field 3	Field 4	Field 5	Field 6	Field 7	Field 8	Field 9	Field 10
FTSET	setnam	nfuncs							
FTSET	WING	1							
string	C	I							

Field	Contents
setnam	Name of the fault tree set (max. 8 chars.).
nfuncs	Number of functions in the set (int. > 0).

Fault Tree Data File Card: FTFUNC

Description:

The FTFUNC card allows the user to specify a name for the function, how many systems it contains, what Boolean operation links the functions, an optional field that specifies the number of failures needed for a fault out of the total elements in an AND operation, and a color identification number for highlighting purposes. Each function represents the top level of a particular function being modeled in the tree.

Format, Example, and Data Type:

Field 1	Field 2	Field 3	Field 4	Field 5	Field 6	Field 7	Field 8	Field 9	Field 10
FTFUNC	fncnam	nsys	op	M/N	cbid				
FTFUNC	FLAP 1	2	AND	2/2	0				
string	C	I	C	C	I				

Field	Contents
fncnam	Name of the function (max. 8 chars.).
nsys	Number of systems in the function (int. > 0).
op	Boolean operation for the function. (AND, OR, or blank).
M/N	Number of faults needed for a failure in an AND operation directly under the function.
cbid	Color Bar ID of highlight color for the function (0 < integer < 31).

Fault Tree Data File Card: FTSYS

Description:

The FTSYS card allows the user to specify a name for the system, how many subsystems it contains, what Boolean operation links the systems, and an optional field that specifies the number of failures needed for a fault out of the total elements in an AND operation.

Format, Example, and Data Type:

Field 1	Field 2	Field 3	Field 4	Field 5	Field 6	Field 7	Field 8	Field 9	Field 10
FTSYS	sysnam	nsubs	op	M/N					
FTSYS	HYDR	3	OR						
string	C	I	C	C					

Field	Contents
sysnam	Name of the system (max. 8 chars.).
nsubs	Number of subsystems in the system (int. > 0).
op	Boolean operation for the function. (AND, OR, or blank).
M/N	Number of faults needed for a failure in an AND operation directly under the system.

Fault Tree Data File Card: FTSUB

Description:

The FTSUB card allows the user to specify a name for the subsystem, how many subsystems it contains, what Boolean operation links the subsystems, and an optional field that specifies the number of failures needed for a fault out of the total elements in an AND operation. The FTSUB card is unique in that it is the only card that can have other cards of the same type under it, i.e., the FTSUB cards can be nested several layers deep, while the FTSET, FTFUNC, and FTSYS cards can only appear on their respective levels.

Format, Example, and Data Type:

Field 1	Field 2	Field 3	Field 4	Field 5	Field 6	Field 7	Field 8	Field 9	Field 10
FTSUB	subnam	nitems	op	M/N					
FTSUB	HYD 1	3	AND	3/3					
string	C	I	C	C					

Field	Contents
subnam	Name of the system (max. 8 chars.).
nitems	Number of subsystems or components in the system (int. > 0).
op	Boolean operation for the function. (AND, OR, or blank).
M/N	Number of faults needed for a failure in an AND operation directly under the system.

Fault Tree Data File Card: FTCOMP

Description:

The FTCOMP card allows the user to specify a name for the component, how many target description components it contains, and what Boolean operation links the components.

Format, Example, and Data Type:

Field 1	Field 2	Field 3	Field 4	Field 5	Field 6	Field 7	Field 8	Field 9	Field 10
FTCOMP	cmpnam	ncomps	op						
FTCOMP	LINE 5	5	OR						
string	C	I	C						

Field	Contents
cmpnam	Name of the component (max. 8 chars.).
nitems	Number of (target description) components in the component (int. > 0).
op	Boolean operation for the component. (AND, OR, or blank).

Fault Tree Data File Card: $CNAME

Description:

The $CNAME card allows the user to specify the target description group and component numbers, the P(K) table identification number, the kill type, and a 40-character text description of the component.

Format, Example, and Data Type:

Field 1	Field 2	Field 3	Field 4	Field 5	Field 6	Field 7	Field 8	Field 9	Field 10
$CNAME	gr	co	pknum	ktype	text				
$CNAME	3	5	1	K	Hydraul	pump	#2		
string	C	I	I	C	40-char	string			

Field	Contents
gr	Group number (0 < int. < 12).
co	Component number (1 <= int. <= 999).
pknum	P(K) table number (int. > 0).
ktype	Kill type of the component (e.g., KK, K, A, B, C; max. 8 chars.).
text	Text description of the component (max. 40 chars.).

Appendix A: Bulk Data File Example

```
$COMMENT=============================================================================
$COMMENT        Bulk Data Format Example testall.bdf
$COMMENT        Author: Modified by Brett Grimes
$COMMENT        Date:?
$COMMENT=============================================================================
VEHICLE   TARGET
$COMMENT  THIS TARGET DESCRIPTION IS GENERATED FROM THE SAMPLE
$COMMENT  CASES BUILT FOR THE FASTGEN4 SOFTWARE CHECK OUT.  THE
$COMMENT  SAMPLE CASES WERE MODIFIED SO THAT GID'S AND EID'S
$COMMENT  ARE UNIQUE.  ADDITIONALLY, THE LOCATION OF ELEMENTS WERE
$COMMENT  MOVED SO THAT THE ELEMENTS WOULD NOT OVERLAP EACH OTHER.
$COMMENT
$COMMENT    THIS IS THE BULK DATA DECK FOR A LINE
$COMMENT12345678123456781234567812345678123456781234567812345678123456781234567812345
$COMMENT       2       3       4       5       6       7       8
SECTION        0       1       1       4
GRID           1          -20.00   10.00    5.00
GRID           2          -20.00   10.00   10.00
GRID           3          -20.00   10.00   15.00
GRID           4          -20.00   10.00   20.00
CLINE         01      55       1       2                   .20    0.99
CLINE         02      56       2       3                   .20    0.99
CLINE         03      57       3       4                   .20    0.99
$COMMENT12345678123456781234567812345678123456781234567812345678123456781234567812345
$COMMENT    THIS IS THE BULK DATA DECK FOR A LINE
$COMMENT       2       3       4       5       6       7       8
SECTION        0       2       2       4
GRID          11          -16.00   10.00    5.00
GRID          12          -16.00   10.00   10.00
GRID          13          -16.00   10.00   15.00
GRID          14          -16.00   10.00   20.00
CLINE         11      55      11      12                   .00    0.99
CLINE         12      56      12      13                   .00    0.99
CLINE         13      57      13      14                   .00    0.99
$COMMENT    THIS IS THE BULK DATA DECK FOR A LINE THREAT RADIUS TEST
$COMMENT12345678123456781234567812345678123456781234567812345678123456781234567812345
$COMMENT       2       3       4       5       6       7       8
SECTION        0       3       1       4
GRID          21          -30.00   10.00    5.00
GRID          22          -30.00   10.00   10.00
GRID          23          -30.00   10.00   15.00
GRID          24          -30.00   10.00   20.00
CLINE         21      21      21      22                   .00    0.99
CLINE         22      22      22      23                   .00    0.99
CLINE         23      23      23      24                   .00    0.99
$COMMENT    THIS IS THE BULK DATA DECK FOR A LINE THREAT RADIUS TEST
$COMMENT12345678123456781234567812345678123456781234567812345678123456781234567812345
$COMMENT       2       3       4       5       6       7       8
SECTION        0       4       2       4
```

```
GRID          31            -0.00   10.00    5.00
GRID          32            -0.00   10.00   10.00
GRID          33            -0.00   10.00   15.00
GRID          34            -0.00   10.00   20.00
CLINE         31       21       31      32                      .00    0.99
CLINE         32       22       32      33                      .00    0.99
CLINE         33       23       33      34                      .00    0.99
$COMMENT12345678123456781234567812345678123456781234567812345678123456781234567812345678
$COMMENT       2        3        4       5        6        7        8
SECTION        1        1        1       4
GRID          41           +3.00   07.00   05.00
GRID          42           +3.00   07.00   20.00
GRID          43           +3.00   13.00   20.00
GRID          44           +3.00   13.00   05.00
CQUAD         41       41       41      42       43       44      .15       2
$COMMENT12345678123456781234567812345678123456781234567812345678123456781234567812345678
$COMMENT       2        3        4       5        6        7        8
SECTION        1        2        1       4
GRID          51           +9.00   07.00   05.00
GRID          52           +9.00   07.00   20.00
GRID          53           +9.00   13.00   20.00
GRID          54           +9.00   13.00   05.00
CQUAD         51       41       51      52       53       54      .15       2
$COMMENT12345678123456781234567812345678123456781234567812345678123456781234567812345678
$COMMENT       2        3        4       5        6        7        8        9
SECTION        2        1        1       4
GRID          61           15.00   00.00   00.00
GRID          62           15.00   00.00   20.00
GRID          63           15.00   20.00   20.00
GRID          64           15.00   20.00   00.00
GRID          65           40.00   00.00   00.00
GRID          66           40.00   00.00   20.00
GRID          67           40.00   20.00   20.00
GRID          68           40.00   20.00   00.00
CHEX1         61       51       61      62       63       64       65      6661
61            67       68                                         .05       2
$COMMENT    THIS IS THE BULK DATA DECK FOR A HEXAHEDRON
$COMMENT12345678123456781234567812345678123456781234567812345678123456781234567812345678
$COMMENT       2        3        4       5        6        7        8        9
SECTION        2        2        2       4
GRID          71           45.00   00.00   00.00
GRID          72           45.00   00.00   20.00
GRID          73           45.00   20.00   20.00
GRID          74           45.00   20.00   00.00
GRID          75           60.00   00.00   00.00
GRID          76           60.00   00.00   20.00
GRID          77           60.00   20.00   20.00
GRID          78           60.00   20.00   00.00
CHEX1         71       51       71      72       73       74       75      7631
31            77       78                                         .00       1
$COMMENT    THIS IS THE BULK DATA DECK FOR A HEXAHEDRON
$COMMENT12345678123456781234567812345678123456781234567812345678123456781234567812345678
$COMMENT       2        3        4       5        6        7        8        9
```

```
SECTION        2       3       2       4
GRID          81            75.00   00.00   00.00
GRID          82            75.00   00.00   20.00
GRID          83            75.00   20.00   20.00
GRID          84            75.00   20.00   00.00
GRID          85            95.00   00.00   00.00
GRID          86            95.00   00.00   20.00
GRID          87            95.00   20.00   20.00
GRID          88            95.00   20.00   00.00
CHEX2         81      51      81      82      83      84      85      8631
31            87      88
$COMMENT    THIS IS THE BULK DATA DECK FOR A THICK PLATE
$COMMENT12345678123456781234567812345678123456781234567812345678123456781234567812345678
$COMMENT       2       3       4       5       6       7       8       9
SECTION        2       4       2       4
GRID          91           105.00   00.00   00.00
GRID          92           105.00   00.00   20.00
GRID          93           105.00   20.00   20.00
GRID          94           105.00   20.00   00.00
GRID          95           120.00   00.00   00.00
GRID          96           120.00   00.00   20.00
GRID          97           120.00   20.00   20.00
GRID          98           120.00   20.00   00.00
GRID          99           140.00   00.00   00.00
GRID         110           140.00   00.00   20.00
GRID         111           140.00   20.00   20.00
GRID         112           140.00   20.00   00.00
CHEX2         91      51      91      92      93      94      95      9631
31            97      98
CHEX2         92      51      95      96      97      98      99     11032
32           111     112
$COMMENT    THIS IS THE BULK DATA DECK FOR A TRIANGLE
$COMMENT12345678123456781234567812345678123456781234567812345678123456781234567812345
$COMMENT       2       3       4       5       6       7       8
SECTION        3       1       1       4
GRID         101           155.00   00.00   00.00
GRID         102           155.00   00.00   20.00
GRID         103           155.00   20.00   20.00
GRID         104           155.00   20.00   00.00
CTRI         101      51     101     102     103             .05       2
CTRI         102      52     101     103     104             .05       2
$COMMENT    THIS IS THE BULK DATA DECK FOR A QUADRILATERAL
$COMMENT12345678123456781234567812345678123456781234567812345678123456781234567812345
$COMMENT       2       3       4       5       6       7       8
SECTION        3       2       1       4
GRID         111           165.00   00.00   00.00
GRID         112           165.00   00.00   20.00
GRID         113           165.00   20.00   20.00
GRID         114           165.00   20.00   00.00
CQUAD        111      51     111     112     113     114     .05       2
$COMMENT   THIS IS THE BULK DATA DECK FOR A SPHERE
$COMMENT12345678123456781234567812345678123456781234567812345678123456781234567812345
$COMMENT       2       3       4       5       6       7       8
```

```
SECTION       10       1       1       4
GRID         117          200.00   00.00   00.00
CSPHERE      117      51     117                            5.0     10.0
$COMMENT   THIS IS THE BULK DATA DECK FOR A SPHERE
$COMMENT12345678123456781234567812345678123456781234567812345678123456781234567812345
$COMMENT       2       3       4       5       6       7       8
SECTION       10       2       2       4
GRID         118          230.00   00.00   00.00
CSPHERE      118      51     118                            5.0     10.0
$COMMENT   THIS IS THE BULK DATA DECK FOR A THIN WALL CONE
$COMMENT12345678123456781234567812345678123456781234567812345678123456781234567812345
$COMMENT       2       3       4       5       6       7       8
SECTION        4       3       1       4
GRID         126          260.00   00.00   10.00
GRID         127          290.00   00.00   10.00
CCONE1       126      65     127     126                    .05     05.031
31          10.0       1       1
$COMMENT   THIS IS THE BULK DATA DECK FOR A THIN WALL CONE
$COMMENT12345678123456781234567812345678123456781234567812345678123456781234567812345
$COMMENT       2       3       4       5       6       7       8
SECTION        4       4       1       4
GRID         128          330.00   00.00   00.00
GRID         129          330.00   00.00   50.00
CCONE1       128      65     128     129                    .05     05.031
31          10.0       2       2
$COMMENT   THIS IS THE BULK DATA DECK FOR A THIN WALL CONE
$COMMENT12345678123456781234567812345678123456781234567812345678123456781234567812345
$COMMENT       2       3       4       5       6       7       8
SECTION        4       5       2       4
GRID         136          370.00   00.00  -10.00
GRID         137          340.00   00.00  -10.00
CCONE1       136      65     137     136                    .00     05.031
31          10.0       2       2
$COMMENT   THIS IS THE BULK DATA DECK FOR A THICK WALL CONE
$COMMENT12345678123456781234567812345678123456781234567812345678123456781234567812345
$COMMENT       2       3       4       5       6       7       8
SECTION        4       6       2       4
GRID         146          430.00   00.00   00.00
GRID         147          400.00   00.00   00.00
CCONE2       146      65     147     146                            05.031
31          10.0     03.     07.
$COMMENT   THIS IS THE BULK DATA DECK FOR VOLUME SUBTRACTION
HOLE           4       8       4     998
$COMMENT12345678123456781234567812345678123456781234567812345678123456781234567812345
$COMMENT       2       3       4       5       6       7       8
SECTION        4     998       2       4
GRID         156          490.00   00.00   00.00
GRID         157          460.00   00.00   00.00
CCONE1       156      62     157     156                    .00     03.031
31          07.0       2       2
$COMMENT12345678123456781234567812345678123456781234567812345678123456781234567812345
$COMMENT       2       3       4       5       6       7       8
SECTION        4       8       2       4
```

```
GRID          166           490.00   00.00   00.00
GRID          167           460.00   00.00   00.00
CCONE1        166       61      167      166                      .00     05.031
31           10.0        2        2
$COMMENT    THIS IS THE BULK DATA DECK FOR VOLUME INTERFERENCE
WALL            4        9        4      999
$COMMENT12345678123456781234567812345678123456781234567812345678123456781234567812345
$COMMENT        2        3        4        5        6        7        8
SECTION         4        9        2        4
GRID          176           560.00   00.00   00.00
GRID          177           530.00   00.00   00.00
CCONE1        176       61      177      176                      .00     05.031
31           10.0        2        2
$COMMENT12345678123456781234567812345678123456781234567812345678123456781234567812345
$COMMENT        2        3        4        5        6        7        8
SECTION         4      999        2        4
GRID          186           559.00   00.00   00.00
GRID          187           531.00   00.00   00.00
CCONE1        186       62      187      186                      .00     04.031
31           09.0        2        2
$COMMENT    THIS IS THE BULK DATA DECK FOR VOLUME SUBTRACTION
HOLE            5        1        5        2
$COMMENT12345678123456781234567812345678123456781234567812345678123456781234567812345
$COMMENT        2        3        4        5        6        7        8
SECTION         5        1        2        4
GRID          211           605.00   00.00   00.00
GRID          212           605.00   00.00   20.00
GRID          213           605.00   20.00   00.00
GRID          241           620.00   00.00   00.00
GRID          242           620.00   00.00   20.00
GRID          243           620.00   20.00   00.00
CTRI          211       18      213      212      211              .00        1
CTRI          212       48      243      242      241              .00        1
$COMMENT12345678123456781234567812345678123456781234567812345678123456781234567812345
$COMMENT        2        3        4        5        6        7        8
SECTION         5        2        2        4
GRID          221           610.00   00.00   00.00
GRID          222           610.00   00.00   20.00
GRID          223           610.00   20.00   00.00
GRID          231           615.00   00.00   00.00
GRID          232           615.00   00.00   20.00
GRID          233           615.00   20.00   00.00
CTRI          221       28      223      222      221              .00        1
CTRI          222       38      233      232      231              .00        1
$COMMENT  THIS IS THE BULK DATA DECK FOR VOLUME INTERFERENCE
WALL            5        5        5        6
$COMMENT12345678123456781234567812345678123456781234567812345678123456781234567812345
$COMMENT        2        3        4        5        6        7        8
SECTION         5        5        2        4
GRID          311           655.00   00.00   00.00
GRID          312           655.00   00.00   20.00
GRID          313           655.00   20.00   00.00
GRID          341           670.00   00.00   00.00
```

```
GRID          342           670.00   00.00   20.00
GRID          343           670.00   20.00   00.00
CTRI          311      18     313      312     311              .00       1
CTRI          312      48     343      342     341              .00       1
$COMMENT12345678123456781234567812345678123456781234567812345678123456781234567812345
$COMMENT        2        3        4        5        6        7        8
SECTION         5        6        2        4
GRID          321           660.00   00.00   00.00
GRID          322           660.00   00.00   20.00
GRID          323           660.00   20.00   00.00
GRID          331           665.00   00.00   00.00
GRID          332           665.00   00.00   20.00
GRID          333           665.00   20.00   00.00
CTRI          321      28     323      322     321              .00       1
CTRI          332      38     333      332     331              .00       1
$COMMENT    THIS IS THE BULK DATA DECK FOR VOLUME SUBTRACTION
HOLE            6        1        6        2
$COMMENT12345678123456781234567812345678123456781234567812345678123456781234567812345
$COMMENT        2        3        4        5        6        7        8
SECTION         6        1        2        4
GRID          411           700.00   00.00   00.00
GRID          412           700.00   00.00   20.00
GRID          413           700.00   20.00   20.00
GRID          414           700.00   20.00   00.00
GRID          441           720.00   00.00   00.00
GRID          442           720.00   00.00   20.00
GRID          443           720.00   20.00   20.00
GRID          444           720.00   20.00   00.00
CHEX2         411      18     411      412     413     414     441     44219
19            443     444
$COMMENT123456781234567812345678123456781234567812345678123456781234567812345678
$COMMENT        2        3        4        5        6        7        8
SECTION         6        2        2        4
GRID          421           705.00   05.00   05.00
GRID          422           705.00   05.00   15.00
GRID          423           705.00   15.00   15.00
GRID          424           705.00   15.00   05.00
GRID          431           715.00   05.00   05.00
GRID          432           715.00   05.00   15.00
GRID          433           715.00   15.00   15.00
GRID          434           715.00   15.00   05.00
CQUAD         421      28     421      422     423     424     .00       2
CQUAD         422      38     431      432     433     434     .00       2
CQUAD         423      48     421      431     434     424     .00       2
CQUAD         424      58     422      432     433     423     .00       2
CQUAD         425      68     421      431     432     422     .00       2
CQUAD         426      78     424      434     433     423     .00       2
$COMMENT    THIS IS THE BULK DATA DECK FOR VOLUME INTERFERENCE
WALL            7        1        7        2
$COMMENT12345678123456781234567812345678123456781234567812345678123456781234567812345
$COMMENT        2        3        4        5        6        7        8
SECTION         7        1        2        4
GRID          511           800.00   00.00   00.00
```

```
GRID          512           800.00   00.00   20.00
GRID          513           800.00   20.00   20.00
GRID          514           800.00   20.00   00.00
GRID          541           820.00   00.00   00.00
GRID          542           820.00   00.00   20.00
GRID          543           820.00   20.00   20.00
GRID          544           820.00   20.00   00.00
CHEX2         511      18     511     512     513     514     541     54219
19            543     544
$COMMENT12345678123456781234567812345678123456781234567812345678123456781234567812345678
$COMMENT       2       3       4       5       6       7       8
SECTION        7       2       2       4
GRID          521           805.00   05.00   05.00
GRID          522           805.00   05.00   15.00
GRID          523           805.00   15.00   15.00
GRID          524           805.00   15.00   05.00
GRID          531           815.00   05.00   05.00
GRID          532           815.00   05.00   15.00
GRID          533           815.00   15.00   15.00
GRID          534           815.00   15.00   05.00
CQUAD         521      28     521     522     523     524     .00       2
CQUAD         522      38     531     532     533     534     .00       2
CQUAD         523      48     521     531     534     524     .00       2
CQUAD         524      58     522     532     533     523     .00       2
CQUAD         525      68     521     531     532     522     .00       2
CQUAD         526      78     524     534     533     523     .00       2
$COMMENT  THIS IS THE BULK DATA DECK FOR VOLUME INTERFERENCE
WALL           8       1       8       2
WALL           9       1       9       2
$COMMENT12345678123456781234567812345678123456781234567812345678123456781234567812345
$COMMENT       2       3       4       5       6       7       8
SECTION        8       1       2       4
GRID          611           905.00   00.00   00.00
GRID          612           905.00   00.00   20.00
GRID          613           905.00   20.00   00.00
GRID          641           920.00   00.00   00.00
GRID          642           920.00   00.00   20.00
GRID          643           920.00   20.00   00.00
CTRI          611      18     613     612     611             .00       1
CTRI          612      48     643     642     641             .00       1
$COMMENT12345678123456781234567812345678123456781234567812345678123456781234567812345
$COMMENT       2       3       4       5       6       7       8
SECTION        8       2       1       4
GRID          621           910.00   00.00   00.00
GRID          622           910.00   00.00   20.00
GRID          623           910.00   20.00   00.00
CTRI          621      28     623     622     621             .05       1
$COMMENT12345678123456781234567812345678123456781234567812345678123456781234567812345
$COMMENT       2       3       4       5       6       7       8
SECTION        9       1       2       4
GRID          711           905.00   00.00   00.00
GRID          712           905.00   00.00   20.00
GRID          713           905.00   20.00   00.00
```

```
GRID          741           920.00    00.00    00.00
GRID          742           920.00    00.00    20.00
GRID          743           920.00    20.00    00.00
CTRI          711      18     713      712      711              .00       1
CTRI          712      48     743      742      741              .00       1
$COMMENT12345678123456781234567812345678123456781234567812345678123456781234567812345
$COMMENT       2        3        4        5        6        7        8
SECTION        9        2        1        4
GRID          721           910.00    00.00    00.00
GRID          722           910.00    00.00    20.00
GRID          723           910.00    20.00    00.00
CTRI          721      28     723      722      721              .05       1
$COMMENT12345678123456781234567812345678123456781234567812345678123456781234567812345
$COMMENT       2        3        4        5        6        7        8
SECTION       10        2        1        4
GRID          821          1010.00    00.00    00.00
GRID          822          1010.00    00.00    20.00
GRID          823          1010.00    20.00    00.00
CTRI          821      28     823      822      821              .05       1
$COMMENT12345678123456781234567812345678123456781234567812345678123456781234567812345678
SECTION       11        1        1        1
POINT          1          -324.42    49.47   -93.92        0       20
POINT          2            94.01   202.43   140.48        0       20
POINT          3             3.70   199.31   218.95        0       20
POINT          4            11.50  -276.61  -113.54        0       20
POINT          5           -15.78   219.23   197.89        0       20
POINT          6            76.32  -289.00    -9.22        0       20
POINT          7          -344.40  -107.07   -44.40        0       20
POINT          8            34.02   180.37   227.41        0       20
POINT          9          -242.73  -245.33   -73.22        0       20
POINT         10          -307.86  -185.04   -49.12        0       20
POINT         11          -324.32    96.59   -49.93        0       20
POINT         12           205.09   -60.56   161.47        0       20
POINT         13          -351.94  -100.43   -21.60        0       20
POINT         14           146.86  -238.60    64.90        0       20
POINT         15          -211.94   231.44    66.56        0       20
ENDDATA
```

Appendix B: Contour Data File Example

```
$COMMENT=============================================================================
$COMMENT        Contour Data Format Example ae.cdf
$COMMENT        Author: Brett F. Grimes
$COMMENT        Date:?
$COMMENT=============================================================================
$COMMENT12345678123456781234567812345678123456781234567812345678
LIMITS          1     0.1     0.0
LIMITS          2     0.2     0.1
LIMITS          3     0.3     0.2
LIMITS          4     0.4     0.3
LIMITS          5     0.5     0.4
LIMITS          6     0.6     0.5
LIMITS          7     0.7     0.6
LIMITS          8     0.8     0.7
LIMITS          9     0.9     0.8
LIMITS         10     1.0     0.9
TMSTEPS         2
$COMMENT VIEW cards
VIEW                  555     444    1.00     5.0
POINT         444       1    0.00    0.00    0.00
POINT         555       1  110.00  110.00  110.00
VIEW                  888     777    2.00    15.0
POINT         777       1    0.00    0.00    0.00
POINT         888       1  110.00 -110.00  110.00
$COMMENT CD cards
CD              0       1    5001    1.00    0.24
CD              0       1    5001    2.00    0.54
ENDDATA
```

Appendix C: Group Defaults File Example

```
$COMMENT=============================================================================
$COMMENT        Group Defaults Format Example ep.gdf
$COMMENT        Author: Brett F. Grimes
$COMMENT        Date:?
$COMMENT=============================================================================
$COMMENT12345678123456781234567812345678123456781234567812345678
COLORGR         0       0
GRPNAME         0  SAUCER SECTION
COLORGR         1      26
GRPNAME         1   WARP NACELLES
COLORGR         7      21
GRPNAME         7       MAIN HULL
ENDDATA
```

Appendix D: Fault Tree Data File Example

```
$COMMENT=================================================================================
$COMMENT         Fault Tree Data Format Example pitchn02a.fdf
$COMMENT         Author: Brett F. Grimes
$COMMENT         Date:11/94
$COMMENT=================================================================================
$COMMENT12345678123456781234567812345678123456781234567812345678123456781234567812345678
FTSET    B1BPITCH        1
FTFUNC   PITCHN02        1      AND      30
FTSYS     PITN02R        2      AND
FTSUB    P&RCNT12        2       OR
FTCOMP   P&RCONT1        1      AND
$CNAME          3       51        3051      PITCH/ROLL CONTRLR 2 ELECT
FTCOMP   P&RCONT2        1      AND
$CNAME          3       52        3052      PITCH/ROLL CONTRLR 2 ELECT
$COMMENT=================================================================================
$COMMENT12345678123456781234567812345678123456781234567812345678123456781234567812345678
FTSUB        ENGR        2       OR
FTSUB       ENGRR        4      AND
FTCOMP   PITCHSV1        2       OR
$CNAME          3      210        3210      SCAS CYLINDER LINK 1
$CNAME          3      903        3903      INBOARD PITCH/ROLL SCAS (#1)
FTSUB       ENG34        2       OR
FTCOMP       ENG3        9      AND
$CNAME          1      410        1410      FAN ENGINE #3
$CNAME          1      430        1430      COMPRESSOR
$CNAME          1      450        1450      COMBUSTOR
$CNAME          1      470        1470      TURBINE
$CNAME          1      490        1490      TURBINE FRAME
$CNAME          1      510        1510      AUGMENTOR
$CNAME          1      530        1530      ACCESSORY GEAR BOX
$CNAME          1      551        1551      MIXER
$CNAME          1      552        1552      NOZZLE
FTCOMP       ENG4        9      AND
$CNAME          1      610        1610      FAN ENGINE #4
$CNAME          1      630        1630      COMPRESSOR
$CNAME          1      650        1650      COMBUSTOR
$CNAME          1      670        1670      TURBINE
$CNAME          1      690        1690      TURBINE FRAME
$CNAME          1      710        1610      AUGMENTOR
$CNAME          1      730        1630      ACCESSORY GEAR BOX
$CNAME          1      751        1651      MIXER
$CNAME          1      752        1652      NOZZLE
FTCOMP     HYD3SV        5       OR
$CNAME          3      640        3640      RESERVOIR SYST III
$CNAME          3      641        3641      MASTER PUMP SYST III
$CNAME          3      642        3642      SLAVE PUMP SYST III
$CNAME          3      643        3643      FILTER/TRANSDUCER, SYST III
$CNAME          3      644        3644      N2 BOTTLE SYST III
```

```
FTCOMP    HEATEX3        1       OR
$CNAME          6      723        6723      HEAT EXCHANGER HYD TO FHS LH INTERIOR
$COMMENT=============================================================================
$COMMENT12345678123456781234567812345678123456781234567812345678123456781234567812345678
FTSUB       ENGRL        5      AND
FTCOMP   OBPITROL        1       OR
$CNAME          3      904        3904      INBOARD PITCH/ROLL SCAS (#3)
FTCOMP   PITCHSV2        1       OR
$CNAME          3      211        3211      SCAS CYLINDER LINK 2
FTSUB       ENG12        2       OR
FTCOMP       ENG1        9      AND
$CNAME          1       10        1010      FAN ENGINE #1
$CNAME          1       30        1030      COMPRESSOR
$CNAME          1       50        1050      COMBUSTOR
$CNAME          1       70        1070      TURBINE
$CNAME          1       90        1090      TURBINE FRAME
$CNAME          1      110        1110      AUGMENTOR
$CNAME          1      130        1130      ACCESSORY GEAR BOX
$CNAME          1      151        1151      MIXER
$CNAME          1      152        1152      NOZZLE
FTCOMP       ENG2        9      AND
$CNAME          1      210        1210      FAN ENGINE #2
$CNAME          1      230        1230      COMPRESSOR
$CNAME          1      250        1250      COMBUSTOR
$CNAME          1      270        1270      TURBINE
$CNAME          1      290        1290      TURBINE FRAME
$CNAME          1      310        1310      AUGMENTOR
$CNAME          1      330        1330      ACCESSORY GEAR BOX
$CNAME          1      351        1351      MIXER
$CNAME          1      352        1352      NOZZLE
FTCOMP     HYD1SV        5       OR
$CNAME          3      436        3436      RESERVOIR SYST I
$CNAME          3      437        3437      MASTER PUMP SYST I
$CNAME          3      438        3438      SLAVE PUMP SYST I
$CNAME          3      439        3439      FILTER/TRANSDUCER, SYST I
$CNAME          3      440        3440      N2 BOTTLE SYST I
FTCOMP    HEATEX1        1       OR
$CNAME          6      729        6729      HEAT EXCHANGER HYD TO FHS RH INTERIOR
$COMMENT=============================================================================
ENDDATA
```

Appendix E: Fault Tree Output File Example

```
ABCDEF01
        ABCDE0 2/2  <AND>
                ABC0    <AND>
                        A0      <OR>
                                0_1  component A0              , PKNUM =   100, Kill Type = K
                        B0      <OR>
                                0_2  component B0              , PKNUM =   100, Kill Type = K
                        C0      <OR>
                                0_3  component C0              , PKNUM =   100, Kill Type = K
                DEF0 2/2  <AND>
                        D0      <AND>
                                0_4  component D0              , PKNUM =   100, Kill Type = K
                        E0      <AND>
                                0_5  component E0              , PKNUM =   100, Kill Type = K
                        F0      <AND>
                                0_6  component F0              , PKNUM =   100, Kill Type = K
        ABCDE1 2/2  <AND>
                ABC1    <AND>
                        A1      <OR>
                                1_1  component A1              , PKNUM =   100, Kill Type = K
                        B1      <OR>
                                1_2  component B1              , PKNUM =   100, Kill Type = K
                        C1      <OR>
                                1_3  component C1              , PKNUM =   100, Kill Type = K
                DEF1 2/2  <AND>
                        D1      <AND>
                                1_4  component D1              , PKNUM =   100, Kill Type = K
                        E1      <AND>
                                1_5  component E1              , PKNUM =   100, Kill Type = K
                        F1      <AND>
                                1_6  component F1              , PKNUM =   100, Kill Type = K
```

Bibliography

1. Vice, John, "Surviving Desert Storm," Aerospace America, 30: 33,62, August 1992

2. Ball, R. E., *The Fundamentals of Aircraft Combat Survivability Analysis and Design*, AIAA, 1985

3. "HOOPS Graphics System Tutorial", Ithaca Software, Inc., Mountain View, CA

4. "HOOPS Reference Manual", Ithaca Software, Inc., Mountain View, CA

5. Dussault, Heather B., "The Evolution and Practical Applications of Failure Modes and Effects Analyses," Rome Air Development Center, Griffis AFB, March 1983.

6. Roach, Lisa K., "Fault Tree Analysis and Extensions of the V/L Process Structure," Army Research Laboratory, Aberdeen Proving Ground, MD, June 1993.

7. Bass, Lewis, Wynholds, Hans, Porterfield III, William, "Logic Tree Analysis - A Fault Tree Graphics Approach," LMSC-D368008 Revision A, Defense Technical Information Center E034-0932, August 1974.

8. Buckland, Michael E., Webster, Mark, Tkalcevic, Frank J., "GRAFTED - GRAphical Fault Tree EDitor: A Fault Tree Description Program for Target Vulnerability/Survivability Analysis - User Manual," DSTO Materials Research Laboratory, Ascot Vale, Victoria, Australia, December 1993

9. Deitz, Paul H., Starks, Michael W., Smith, Jill H., Ozolins, Aivars, "Current Simulation Methods in Military Systems Vulnerability Assessment," Ballistic Research Laboratory, Aberdeen Proving Ground, MD, November 1990

10. Deitz, Paul H., "Computer-Aided Techniques for Survivability/Lethality Modeling," Ballistic Research Laboratory, Aberdeen Proving Ground, MD, February 1988

11. Deitz, Paul H., "Modern Computer-Aided Tools for High-Resolution Weapons System Engineering," Ballistic Research Laboratory, Aberdeen Proving Ground, MD, February 1988

12. Starks, Michael W., "Assessing the Accuracy of Vulnerability Models by Comparison with Vulnerability Experiments," Ballistic Research Laboratory, Aberdeen Proving Ground, MD, July 1989

13. Barlow, R. E., Lambert, H. E., "Introduction to Fault Tree Analysis," Reliability and Fault Tree Analysis, SIAM, Philadelphia, PA, 1975, pages 7-35

14. Young, Jonathan, "Using the Fault Tree Analysis Technique," Reliability and Fault Tree Analysis, SIAM, Philadelphia, PA, 1975, pages 827-847

15. Bass, Lewis, Wynholds, Hans W., Porterfield, William R., "Fault Tree Graphics," Reliability and Fault Tree Analysis, SIAM, Philadelphia, PA, 1975, pages 913-927

16. Pascal, Andrew M., Foulk, Jeffery W., Vikestad, Walter S., "Integration of Vulnerability Analysis Requirements Into Aircraft Joint Live Fire (JLF) Test Plans," The SURVICE Engineering Company, Aberdeen, MD, December 1990

17. Klopcic, J. Terrence, "Survey of Vulnerability Methodological Needs," Ballistic Research Laboratory, Aberdeen Proving Ground, MD, November 1991

18. Survivability Evaluation Branch, "F/A-18 Baseline Combat Survivabilty Evaluation, Appendix E: Fault Trees," Survivability and Lethality Division, Naval Weapons Center, China Lake, CA, May 1982

19. Grover, John M., "A Methodology for Determining the Survivability of Fixed-Wing Aircraft Against Small Arms," AFIT/GST/ENS/89M-05, Wright-Patterson AFB, OH, March 1989

20. Kavanagh, Kevin John, "MacIntosh Survivability Assessment Program," Naval Postgraduate School, Monterey, CA, September 1991

21. Gil, Leopoldo Dolano Jr., "A Comparison of Three Computer Weapon-Target Endgame Simulations for Aircraft," Naval Postgraduate School, Monterey, CA, June 1992

22. Bruenning, Leonard E. Jr., Womble, Alyndia S., "High-Explosive Incendiary Vulnerability Assessment Model (HEIVAM)," Volume I: User Manual, Air Force Armament Laboratory, Eglin AFB, FL, September 1986.

23. Bruenning, Leonard E. Jr., Womble, Alyndia S., "High-Explosive Incendiary Vulnerability Assessment Model (HEIVAM)," Volume II: Analyst Manual, Book 1, Air Force Armament Laboratory, Eglin AFB, FL, September 1986.

24. Bennett, Gerald, Crosthwaite, Kevin R., "A Summary of Aerospace Vehicle Computerized Geometric Descriptions for Vulnerability Analyses," ASD-TR-91-5032, Advanced Systems Analysis Directorate, ASD, Wright-Patterson AFB, OH, May 1992

25. "COVART 4.0 User Manual (Draft)," Task Report, KETRON Division of The Bionetics Corporation, Baltimore, MD, ASC/YFEX and ASC/XRESV, Wright-Patterson AFB, OH, April 1994

26. Griffis, Hugh, Lentz, Marty, "FASTGEN 4.1 User's Manual," ASC/XRESV and WL/FIVS, Wright-Patterson AFB, OH, April 1994

Vita

Brett F. Grimes was born on February 5, 1965 in St. Marys, Ohio. He graduated from Memorial High School in St. Marys in 1983 and attended Wright State University in Dayton, Ohio. He graduated in 1989 with a Bachelor of Science in Computer Engineering. Upon graduation, he spent some months in St. Louis before taking a position with the Scientific and Engineering Applications section of the Information Systems Technical Center at Wright-Patterson AFB, Ohio. His duties included evaluating, installing, and maintaining software applications, and eventually defining the guidelines for application management for the organization. He was also responsible for the development of a Unix workstation demonstration laboratory for the evaluation of hardware and software products. Later, he went on to assemble the Wright-Patterson Major Shared Resource Center Visualization Laboratory. Early on however, he became involved with graphics programming projects that eventually led to his thesis efforts in the School of Engineering at the Air Force Institute of Technology.

Permanent Address:

7922-C Moulins Dr.

Centerville, OH 45459